Facultade de Informática

UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

Sistema de representación virtual interactiva del modelo de una estructura arquitectónica de alto valor cultural

Estudiante: Adrián Xuíz García
Dirección: Ángel Gómez García
Estefanía López Salas

A Coruña, febrero de 2020.

Dedicado a mi familia y sobre todo a mis padres, porque sin ellos nada de esto sería posible. Gracias por todo.

Agradecimientos

Quiero agradecer primeramente a mi familia y a mis amigos más cercanos por formarme como la persona que soy hoy en día.

A todos mis compañeros y compañeras de carrera, con quienes sufrí y disfruté todos estos años.

A mis directores de TFG, por su esfuerzo y dedicación en este proyecto.

Y por último, pero no menos importante, a todos los profesores que disfrutan con su trabajo y hacen del proceso de aprender una experiencia bonita y enriquecedora.

Resumen

Los avances tecnológicos en el campo de la representación de entornos virtuales interactivos permiten obtener modelados realistas de estructuras arquitectónicas y emplearlos para diversos ámbitos como la investigación, la educación, la divulgación o el entretenimiento.

En este proyecto se desarrollará una herramienta inmersiva para la representación 3D de un sitio histórico como es el monasterio benedictino gallego de San Julián de Samos, monumento nacional histórico-artístico desde septiembre de 1944 y Bien de Interés Cultural (BIC) que forma parte del Camino de Santiago, con todo el valor que la ruta jacobea tiene por sí misma.

Se partirá de varios modelos del monasterio que serán convertidos y adaptados para construir un entorno virtual interactivo que el usuario podrá visitar escogiendo el recorrido a su voluntad, a la vez que tendrá la posibilidad de interactuar con los elementos que conformen dicho entorno, accediendo a diferentes contenidos seleccionados con la finalidad de permitir un conocimiento histórico del proceso de transformación en el tiempo de este sitio monástico.

La aplicación ha sido implementada en el entorno de desarrollo Unity, y proporciona al usuario la posibilidad de experimentar de forma virtual las diferentes fases evolutivas del sitio histórico, interaccionar con los elementos 3D o apreciar los cambios espaciotemporales, facilitando su comprensión del entorno de manera intuitiva y rigurosa.

Abstract

Technological advances in the field of interactive virtual environment representation allow realistic modeling of architectural structures to be obtained and used for various fields such as research, education, dissemination or entertainment.

This project will develop an immersive tool for the 3D representation of a historic site such as the Galician Benedictine monastery of San Julián de Samos, a national historical-artistic monument since September 1944 and Cultural Interest (BIC) that is part of the Camino Santiago, with all the value that the Jacobean route has for itself.

It will be based on several models of the monastery that will be converted and adapted to build an interactive virtual environment that the user can visit by choosing the route at will,

while having the possibility to interact with the elements that make up that environment, accessing different selected contents with the purpose of allowing a historical knowledge of the transformation process over time of this monastic site.

The application has been implemented in the Unity development environment, and provides the user with the possibility to experience the different evolutionary phases of the historical site, interact with 3D elements or appreciate space-time changes, facilitating their understanding of the environment in an intuitive and rigorous way.

Palabras clave:

cultural

virtual

3D

entorno inmersivo

monasterio Samos

Cultunity3D

Unity

Keywords:

cultural

virtual

3D

immersive environment

monastery of Samos

Cultunity3D

Unity

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este primer capítulo se presentan la motivación, los objetivos del proyecto, y la estructura de la memoria.

1.1 Motivación

En los últimos años los entornos 3D virtuales han tenido un gran auge en sectores como la arquitectura, el cine o los videojuegos. Sin embargo, son pocas las herramientas que ofrecen un sistema basado en esos entornos con el objetivo de informar y educar sobre bienes de interés cultural tales como iglesias, castillos o yacimientos arqueológicos, entre otros.
Aunque es habitual la creación y uso de modelos tridimensionales de bienes culturales por parte de profesionales, investigadores y científicos que trabajan en cuestiones patrimoniales, no es tan fácil encontrar una solución que sea accesible al gran público, no tan habituado a manejar construcciones virtuales y a extraer conocimiento útil de ello sin una preparación previa.

En este sentido, es importante buscar un equilibrio entre los modelos detallados y llamativos de los videojuegos, y los modelos realistas y precisos desde un punto de vista histórico de los profesionales.
Por tanto, el objetivo principal del proyecto será el desarrollo de una aplicación de visualización avanzada basada en entornos virtuales 3D, para lograr un producto accesible y atractivo que ayude a las personas interesadas en ciertas temáticas culturales a informarse con una perspectiva rigurosa y moderna.

Además, se busca que la herramienta a desarrollar sea fácilmente manejable y sencilla de

implementar, para que aquellos que busquen crear una aplicación similar, puedan hacerlo de forma simple, con facilidad de personalización y minimizando los tiempos de desarrollo y coste. Por ello, se ha orientado su desarrollo con el objetivo de ser empleada también por otros profesionales del campo de investigación cultural, no solo por desarrolladores de software.

Este trabajo es un intento de facilitar las labores de creación de entornos de valor educativo. Tiene como objetivo ampliar las enseñanzas culturales al gran público, a la par que dotar de una herramienta de representación y distribución de conocimiento a los expertos.

1.2 Alcance y objetivos

El objetivo principal de este proyecto es crear un entorno de realidad virtual del monasterio gallego de San Julián de Samos, que permita a su futuro usuario experimentar los cambios espacio-temporales de un sitio histórico con una larga trayectoria vital.

Para ello, se determinaron los siguientes objetivos secundarios:

- Dotar al sistema de las funcionalidades necesarias para permitir el movimiento libre y la detección de objetos en el entorno virtual.
- Poder seleccionar y cambiar la época histórica a visualizar.
- Ofrecer al usuario distintos tipos de funcionalidades de interacción con determinados elementos del entorno, para proporcionar información relevante sobre un determinado aspecto.
- Implementar características que permitan mostrar elementos informativos adicionales como imágenes, textos o audios.
- Implementar un núcleo de funcionalidades que simplifique el desarrollo y/o modificación de proyectos similares.

1.3 Estructura de la memoria

Esta memoria se ha estructurado en base a las sucesivas fases del desarrollo del proyecto. A continuación se describe cada uno de los capítulos que la componen:

- **Capítulo 1: Introducción**
  En este capítulo se proporciona un contexto general al lector, se plantean los objetivos principales del proyecto y su motivación, y se explica la estructura de la memoria.

- **Capítulo 2. Tecnologías y herramientas**
  En este capítulo se describen brevemente las tecnologías y herramientas empleadas para la elaboración del proyecto en cada una de sus fases.

- **Capítulo 3: Estado del arte**
  En este capítulo se realiza una búsqueda y revisión sobre el estado actual del campo al que pertenece el proyecto y sobre otros trabajos similares.

- **Capítulo 4: Desarrollo**
  En este capítulo se describen todos los pasos realizados para el desarrollo del proyecto, incluyendo la metodología empleada, las iteraciones en las que se dividió el trabajo, el diseño y la implementación de la herramienta.

- **Capítulo 5: Pruebas**
  En este capítulo se describen las pruebas realizadas a la herramienta para comprobar su correcto funcionamiento.

- **Capítulo 6: Gestión de riesgos**
  En este capítulo se realiza un breve estudio sobre los potenciales riesgos detectados que podrían afectar al proyecto.

- **Capítulo 7: Recursos y costes**
  En este capítulo se realiza un breve estudio sobre los recursos empleados y la estimación de costes del proyecto.

- **Capítulo 8: Conclusiones**
  En este capítulo se describen las conclusiones finales obtenidas al llegar al final del desarrollo del proyecto.

- **Capítulo 9: Líneas futuras**
  En este capítulo se proponen nuevos objetivos o funcionalidades que se podrían implementar en un futuro para ampliar las funcionalidades de la aplicación desarrollada.

Capítulo 2

Tecnologías y herramientas

En este capítulo se describirán brevemente las principales herramientas empleadas en el desarrollo de este proyecto.

2.1 Unity

Figura 2.1: Logotipo oficial de Unity

Dentro del catálogo de entornos de desarrollo 3D, Unity[4] destaca por ser uno de los más potentes y versátiles, y permite su uso gratuito a través de las licencias *Unity Personal*. Siendo realmente un motor de videojuegos multiplataforma, está disponible para los sistemas operativos Windows, Mac OS y GNU/Linux.
Su motor gráfico permite el renderizado mediante OpenGL para todas las plataformas o Direct3D exclusivamente en plataformas Windows. El scripting se realiza a través de Boo, Mono o C#. En este proyecto se optó por utilizar el lenguaje C# para implementar los scripts debido a ser un lenguaje similar a Java. Teniendo en cuenta los conocimientos previos la labor de adaptación a este lenguaje sería más rápida que en los otros casos.
Puesto que es una plataforma que lleva existiendo mucho tiempo (su primera versión se lanzó en 2005 y recibe actualizaciones casi mensuales), es un entorno con amplia variedad de

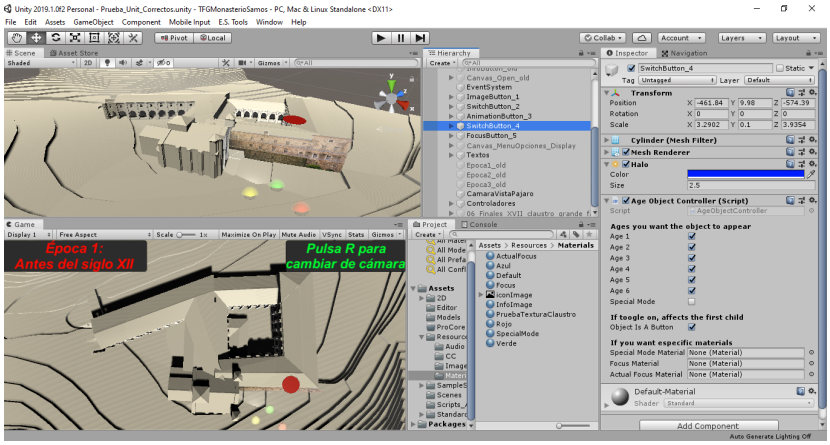


Figura 2.2: Ejemplo de interfaz del editor Unity en el proyecto

contenidos online de ayuda; no sólo por las guías oficiales, sino también por los tutoriales y preguntas/respuestas realizados por los usuarios.
A mayores, tiene muchas características especiales que lo hacen ideal para este trabajo, como la facilidad de importación de los modelos 3D arquitectónicos o el buen acoplamiento de mecanismos externos para facilitar su uso con gafas 3D en entornos virtuales.

2.2 Blender

Figura 2.3: Logo oficial de Blender

Blender[5] es uno de los programas de modelado 3D y animación más extendidos que existen dentro del software libre. Su licencia gratuita, sus funcionalidades profesionales y su fácil manejo lo hacen muy atractivo para ser utilizado junto a un motor gráfico como Unity.

Para este trabajo se utilizó al comienzo del desarrollo para realizar ajustes en los modelos, aunque posteriormente se desechó su uso para ser sustituido por su equivalente con licencia comercial, Autodesk 3D MAX, debido a problemas con la importación de los modelos base.

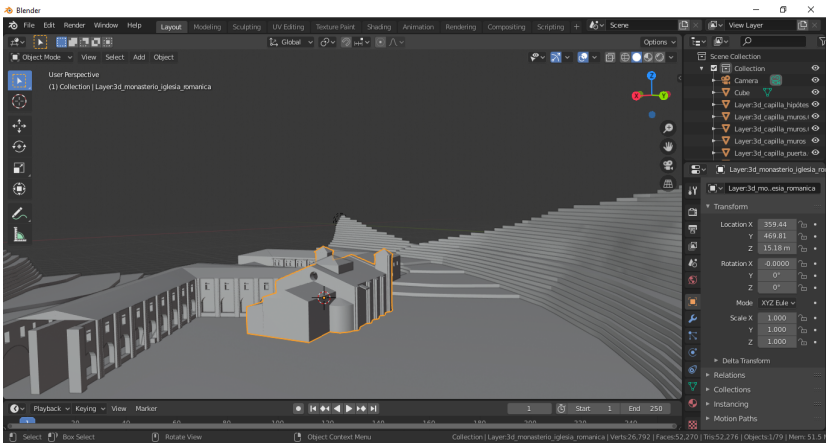


Figura 2.4: Ejemplo de interfaz de Blender

2.3 Autodesk 3D MAX

Figura 2.5: Logo oficial de Autodesk 3D Max

Autodesk 3D Max[6] es un programa comercial para modelado 3D y animación utilizado en distintos sectores como arquitectura, videojuegos, televisión, o cine. Con su tecnología basada en plugins y sus más de 30 años de experiencia, es uno de los líderes del mercado en su campo. Los modelos 3D arquitectónicos utilizados como base para este proyecto realizados por la profesora Estefanía López Salas fueron ajustados con esta aplicación tras su creación. Sobre estos modelos se realizaron diversos cambios y modificaciones en los formatos de los archivos con este programa para conseguir que su importación al entorno Unity fuese lo más sencilla posible.

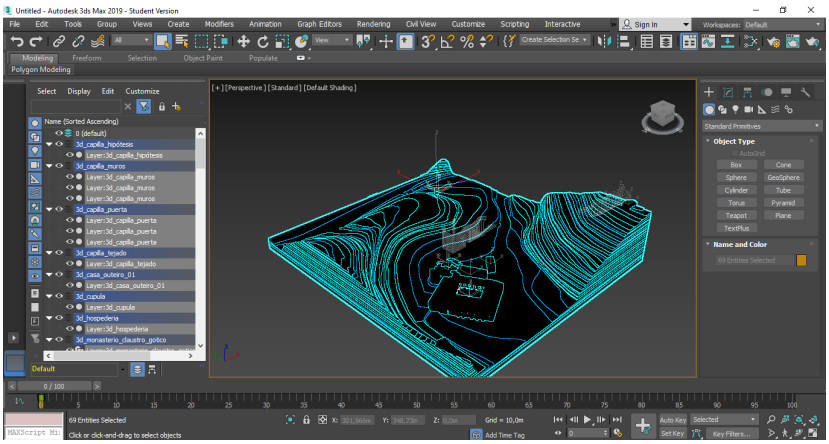


Figura 2.6: Ejemplo de interfaz de Autodesk 3D Max en el proyecto

Para este trabajo se empleó la versión 2019, que dispone de licencias gratuitas para estudiantes.

2.4 Notepad++

Figura 2.7: Logo oficial de Notepad++

Notepad++[7] es un potente editor de texto que proporciona funcionalidades importantes para la implementación de código fuente como es la posibilidad de abrir múltiples pestañas, funciones de Buscar y Reemplazar o Autocompletar, sangrado automático, resaltado de sintaxis de multitud de lenguajes de programación, etc. E incluso es posible añadir más funcionalidades a partir de plugins.

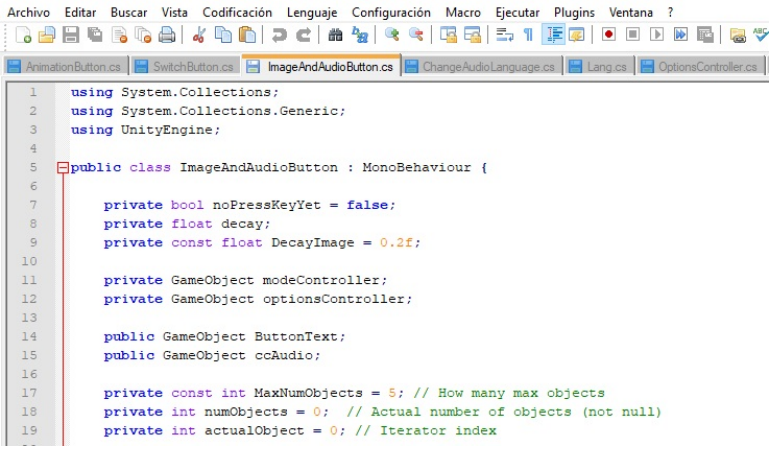


Figura 2.8: Ejemplo de interfaz de Notepad++ con un script del proyecto

Esta herramienta no tiene los beneficios de otros editores gratuitos más modernos como Atom o Visual Studio Code, pero su flexibilidad y sencillez de uso lo hacen perfecto para editar archivos pequeños como son los scripts de este proyecto.

Capítulo 3

Estado del arte

En este capítulo se realiza una breve descripción de los motores gráficos y de dos de sus ejemplos más utilizados. A continuación se describe el panorama actual en los sistemas de visualización de entornos de valor cultural, y se analizan los más relevantes para este proyecto.

3.1 Motores gráficos

Un motor gráfico es un software usado por aplicaciones y programas para dibujar gráficos en la pantalla de nuestro ordenador, dispositivo móvil o tablet. En el mundo del entretenimiento, es una plataforma orientada a producir un videojuego en todas sus fases (diseño, implementación y pruebas). Permite a los desarrolladores no entrar en cuestiones específicas como representación de gráficos, producción de sonidos o colisiones entre elementos (abstracción del hardware); para así poder enfocar la mayor parte de los esfuerzos en el propio videojuego (mecánicas, diseño de niveles, diálogos, etc.).

En el mercado podemos encontrar una amplia variedad de motores gráficos (para juegos de estrategia, simuladores, sistemas de control, etc.), pero para este trabajo los más interesantes son los de primera persona, por ofrecer una experiencia más realista.

Aunque los motores gráficos fueron creados originalmente para reducir los costes de producción de los videojuegos, su flexibilidad y gran aporte de herramientas permite que se puedan emplear también para diversas aplicaciones orientadas a la iteración humana, ya que el producto resultante suele ser cómodo de usar y agradable para el usuario final.

Puesto que existe una gran variedad de mercado, se tuvieron en cuenta los que más se aproximaban al objetivo de este TFG, y que permitiesen su uso en cuanto a funcionalidades, precio de licencia, dificultad de aprendizaje, plataforma orientada, etc.

De todos ellos, dos destacan especialmente por su multifuncionalidad y su gran experiencia en el mercado: Unreal Engine y Source.

3.1.1 Unreal Engine

Figura 3.1: Logotipo oficial de Unreal Engine

Unreal Engine[8] es un motor de juego creado por la compañía Epic Games, siendo presentado por primera vez en el shooter Unreal, en 1998. Su código, basado en C++, ha sido la base de juegos tan importantes como Gears of Wars, Mass Effect, BioShock, Batman: Arkham Asylum, SMITE, Tom Clancy's Raibow Six, Outlast, Rocket League o Fornite. La versión actual (Unreal Engine 4) está diseñada para un gran número de plataformas, tales como PS4, Nintendo Switch, Microsoft Windows, MacOS, GNU/Linux, Oculus Rift o Playstation VR.

Ofrece una licencia gratuita para la versión más reciente y todas sus posteriores actualizaciones, con una pequeña tarifa sobre el beneficio del producto si sus ventas superan los 3000$ de ingresos.

Actualmente se usa tanto en el sector de los videojuegos como en arquitectura, cine, transporte, eventos en directo o simuladores de entrenamiento.

3.1.2 Source

Source[9] es un motor de videojuego desarollado por Valve Corporation, una de las empresas de videojuegos más importantes en el panorama internacional. Famosos títulos como Counter-Strike, Left 4 Dead, Dota 2, Portal o Half-Life se realizaron con este motor gráfico como base.

Figura 3.2: Logotipo oficial de Source

Nació en 2004 con Counter Strike y fue evolucionando lentamente gracias a su sistema de actualizaciones casi automáticas (al estar estrechamente ligado a Steam, una de las plataformas de venta de videojuegos online). Ofrece soporte para Microsoft Windows, MacOS, GNU/Linux, Xbox, PS3 y PS4, entre otros.

Actualmente, Valve prepara una segunda versión del software, que al parecer estará disponible para uso público a partir de marzo de 2020[1].

3.2 Sistemas de visualización de entornos de valor cultural

A pesar de que las construcciones de modelos 3D virtuales son empleadas constantemente, la visualización de entornos de valor cultural es una temática que se encuentra en unas fases todavía muy tempranas. Teniendo en cuenta que la temática de este proyecto es precisamente la visualización inmersiva de este tipo de entornos, era imprescindible comprobar el estado del arte para este ámbito.
Algunos de los proyectos realizados en los últimos años sobre este tema y que sirvieron de inspiración para este trabajo se describen en las siguientes secciones.

3.2.1 SpatioScholar

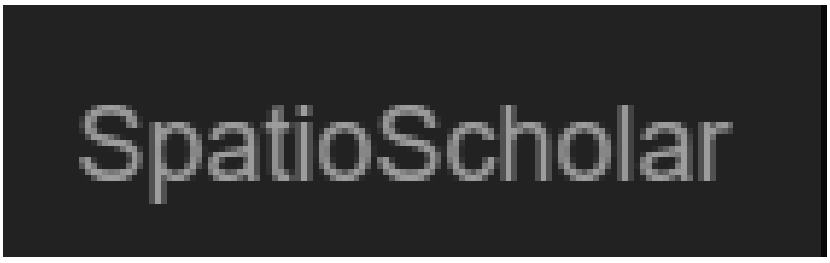


Figura 3.3: Logotipo oficial del SpatioScholar

[1]Fuente: https://www.gamasutra.com/view/news/354549/Valve_releasing_Source_2_tools_alongside_HalfLife_Alyx_next_March.php

SpatioScholar [1] es una plataforma de análisis que permite procesar y visualizar la dimensión temporal de un espacio determinado. El proyecto nació a partir del estudio de la disertación *"Madness and Empire: The Ottoman Asylum, 1830-1930"*[10], de Burcak Ozludil. Con la necesidad de poder explicar cómo se transformaba el asilo otomano a lo largo del tiempo y cómo era la vida en este asilo, Ozludil contactó con Augustus Wendell en 2015 para encontrar una forma de visualizar la información de forma efectiva, para lo cual colaboraron con el estudiante Ulysee Thompson, el cual diseñó e implementó un entorno virtual en Unity. Los tres presentaron su proyecto en 2016 en las conferencias de Digital Humanities y eCAADe[11].

Las características que lo hacen especialmente relevante para este trabajo son las siguientes:

- Se puede cambiar la época de la visualización para apreciar los cambios en el tiempo (Figura 3.4).

Figura 3.4: Vista isométrica que permite cambio de época[1]

- Se puede explorar el entorno en primera persona (Figura 3.5).

- Se pueden ver materiales históricos o contemporáneos usados como base en la investigación (imágenes, fotos, textos,etc.) sobre el modelo o al mismo tiempo que se visualiza el modelo (Figura 3.6).

- Se pueden realizar anotaciones que otros usuarios pueden leer y compartir.

La mayoría de estas funcionalidades fueron consideradas como esenciales para este trabajo, por lo cual se tomaron como inspiración y a partir de ellas se desarrolló una aproximación propia, adaptada a los requisitos de nuestro proyecto.

Figura 3.5: Vista del modelo en primera persona[1]

Figura 3.6: Vista con imagen informativa en ventana emergente[1]

Por desgracia, no se realizaron nuevas actualizaciones en el desarrollo del proyecto desde 2017. Actualmente Ulysee Thompson trabaja en un entorno de realidad virtual para aprender inglés (2019)[2].

Se pueden encontrar varios vídeos representando las características del proyecto disponibles en Internet[3].

[2]Fuente: https://scholar.google.com/citations?user=ezGYNQcAAAAJ&hl
[3]Fuente:https://www.youtube.com/channel/UCtHDnLxNMdhSnzTdYaFfcWQ

3.2.2 UCLA's reconstruction of World Columbian Exposition

Figura 3.7: Logotipo oficial del UST

El Urban Simulation Team (UST) [2] forma parte de la Universidad de California en Los Ángeles (UCLA). Con más de 50 años de historia, su objetivo principal es explorar las diversas aplicaciones existentes para simulación y aplicarlas a diseño urbanístico, planes de emergencia, educación, etc. Es este último apartado el más interesante de cara al trabajo objeto de esta memoria, ya que realizaron diversos proyectos orientados a "recrear y explorar entornos históricos"[4].

Entre los tres proyectos que muestran en su página oficial, el más relevante para este trabajo fue el de la reconstrucción virtual de la World Columbian Exposition (Figura 3.8), realizada en Chicago en 1893.

Figura 3.8: Muestra de la World Columbian Exposition[2]

La World Columbian Exposition fue un hito para el planteamiento urbanístico y la arquitectura norteamericana. A partir de los planos de Burnham & Root y la firma de paisajistas formada por Frederick Law Olmsted y Henry Sargent Codman, un equipo de cinco de las firmas de arquitectos más prestigiosas del país crearon en 1893 una ciudad clásica que tuvo una importante influencia en el devenir de la arquitectura norteamericana posterior, y que empezaría los movimientos de Ciudad Bella y el Renacimiento Americano[5].

[4]Fuente: http://www.ust.ucla.edu/ustweb/projects.html
[5]Fuente: http://www.ust.ucla.edu/ustweb/Projects/columbian_expo.htm

El uso de las tecnologías de simulación virtual a tiempo real permite recrear la experiencia de la exposición de una manera inmersiva, en contraste a emplear únicamente escritos o fotografías estáticas.

La filosofía del UST en muchos de estos proyectos es de monitor-alumnos, en la que una única persona maneja el entorno mientras los demás son sólo testigos. No es lo ideal para este proyecto, pero muchas de las ideas que plantean pueden ser aprovechadas y adaptadas a nuestro caso.

Al igual que sucede con SpatioScholar, el proyecto del UST en lo referente al desarrollo de aplicaciones educativas parece estar detenido. Aunque en la página oficial del UST los últimos cambios oficiales se realizaron en 2013, la presentación más reciente de este proyecto de la que tenemos constancia es de octubre del 2019[6].

[6]Fuente: https://sites.duke.edu/centeringdh/schedule/

Capítulo 4

Desarrollo

En este capítulo se muestra el proceso de desarrollo del proyecto. Se describirá la metodología seleccionada para su realización, y los detalles de su aplicación en las diferentes fases que componen el proceso de análisis, diseño, implementación y pruebas.

4.1 Metodología

4.1.1 Metodología ágil

Cuando hablamos de metodología ágil, estamos hablando de un enfoque de las decisiones tomadas en los proyectos software que se basa en el desarrollo iterativo e incremental[12], donde los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto.

Cada iteración del ciclo de vida incluye planificación, análisis de requisitos, diseño, codificación, pruebas y documentación[12], con el objetivo de implementar sin errores parte de las funcionalidades que se buscan, hasta obtener el producto completo en la última iteración (Figura 4.1).

Frente a la metodología tradicional, la metodología ágil tiene las siguientes ventajas:

- Prevalece el software funcional antes que la documentación comprensiva, lo que es más útil y obtiene mejor *feedback* en las reuniones con clientes.
- Gran adaptación a cambios. Puesto que el desarrollo es continuo y con iteraciones cortas, es sencillo adaptarse a algún cambio de las funcionalidades exigidas.

- Reduce la complejidad del proyecto, al separar el desarrollo en iteraciones.
- El producto es consistente en todas las etapas del desarrollo

Figura 4.1: Esquema de procesos de una metodología ágil

4.1.2 Scrum

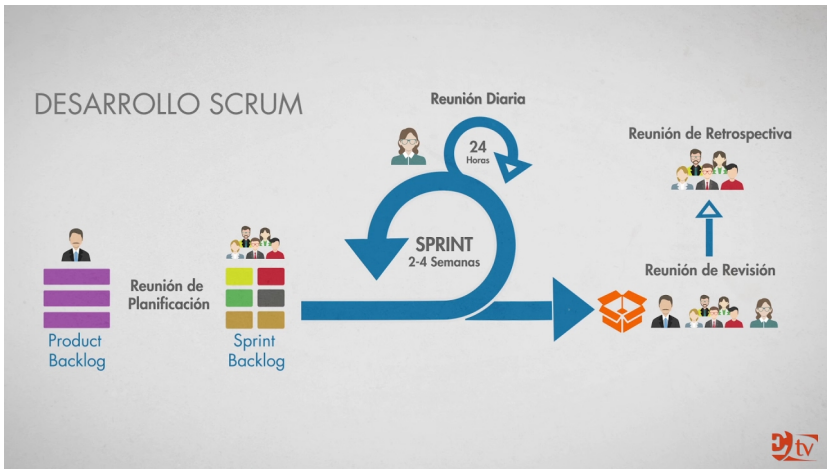


Figura 4.2: Esquema de procesos de Scrum

Dentro de los tipos de metodologías ágiles, uno de los que destaca es Scrum[13], que define una estrategia flexible con un equipo de desarrollo trabajando como una unidad, enfocado en

maximizar sus esfuerzos para adaptarse a los cambios que los clientes pueden necesitar en cualquier momento del desarrollo (Figura 4.2).

4.1.2.1 Sprints

Los sprints o iteraciones son las unidades básicas del desarrollo en Scrum. Consisten en un bloque de tiempo con una duración predeterminada, que se recomienda que esté entre una y cuatro semanas[13]. Cada sprint comienza con la definición de un *Sprint Backlog*, que son los objetivos previstos, y finaliza con un análisis del trabajo realizado y una búsqueda de posibles mejoras para el siguiente sprint. Se realizan diariamente reuniones para planificar el trabajo de día posteriores y controlar el avance del proyecto, informando en caso de que haya obstáculos para intentar solucionarlos lo antes posible.

4.1.2.2 Roles

Dentro de Scrum tenemos 3 roles principales[13]:

- **Scrum Master**: Funciona como un intermediario entre el equipo y los posibles impedimentos que surjan, asegurándose de que el proceso se realiza correctamente.
- **Product Owner**: Es el representante de los clientes y debe comunicarse con ellos y el equipo de desarrollo, estableciendo prioridades y describiendo los deseos de los clientes al equipo pero sin interferir en el proceso técnico del desarrollo.
- **Equipo de desarrollo**: Son equipos pequeños, de 3 a 9 individuos, encargados de realizar los incrementos funcionales del producto en cada iteración. Los equipos se encargan de todas las tareas (analizar, diseñar, implementar, testear, comunicar y documentar) necesarias para el desarrollo del producto.

4.1.2.3 Reuniones

Las reuniones definidas por Scrum son las siguientes[13]:

- **Reunión de planificación**: Se realiza al comienzo de cada Sprint y en ella se determinan los objetivos para la iteración. Tiene una duración máxima de 8 horas.

- **Reunión diaria**: Se realiza diariamente y se hace un pequeño análisis de objetivos cumplidos, problemas encontrados, y objetivos previstos para el día siguiente. Duración máxima de 15 minutos.

- **Reunión de análisis**: Se realiza al final de cada Sprint y en ella se revisa el trabajo realizado y el planificado pero no completado. Duración máxima de 4 horas.

- **Retrospectiva**: Se realiza al final de cada Sprint y en ella los miembros del equipo darán su opinión de puntos mejorables y puntos positivos de la iteración. Es convocada por el Scrum Master.

4.1.2.4 Adaptación de Scrum al proyecto

Scrum está orientado para proyectos de ingeniería en los que los requisitos pueden ir cambiando, como es el caso de este trabajo. Sin embargo, esta metodología está más indicada para organizaciones que dispongan de suficientes recursos humanos y materiales para poder aplicarla de forma estricta.
Por lo tanto, como metodología ágil es adecuado para la planificación de este proyecto, pero requiere modificaciones para poder adaptarlo correctamente, sin perder la filosofía de las metodologías iterativas.

Entre otros ajustes, los sprints se han organizado con una duración aproximada de 4 semanas, ya que no era posible emplear el 100% del tiempo en el proyecto, y se querían implementar aproximadamente dos funcionalidades por Sprint. Con ese tiempo, hay suficiente margen para realizar el diseño, la implementación y las pruebas de cada objetivo.

En cuanto a los roles, los directores asumen el papel de Product Owners, revisando y orientando el proyecto, mientras que el alumno se encargará asumir los roles de Scrum Master y de equipo de desarrollo, implementando y supervisando el proceso de desarrollo de la aplicación.

Para las reuniones, se han eliminado las diarias y las de planificación se verán ajustadas respecto a los objetivos propios de cada iteración. Por lo tanto, su número se reduce, y no se realizarán al final de cada sprint, sino cada dos o tres iteraciones. La reunión de análisis y de retrospectiva se realizarán igualmente, pero con una duración menor.

4.2 Planificación

La planificación de este proyecto se ha realizado siguiendo una metodología ágil e iterativa, Scrum, ajustada a los objetivos y recursos del proyecto. A continuación se muestran los diagramas de Gantt del proyecto, los Sprints o iteraciones realizados y los objetivos buscados.
En los apartados correspondientes al Diseño e Implementación se detallarán las soluciones técnicas utilizadas en el desarrollo del proyecto y que están estrechamente vinculadas con las tareas descritas en este apartado de planificación de las iteraciones.

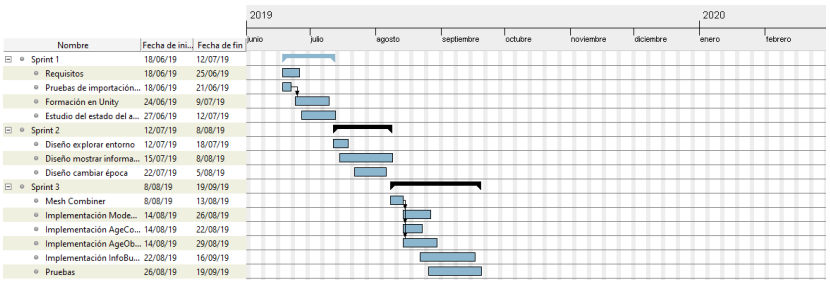


Figura 4.3: Primer diagrama de Gantt de las tareas del proyecto. Creado con GanttProject

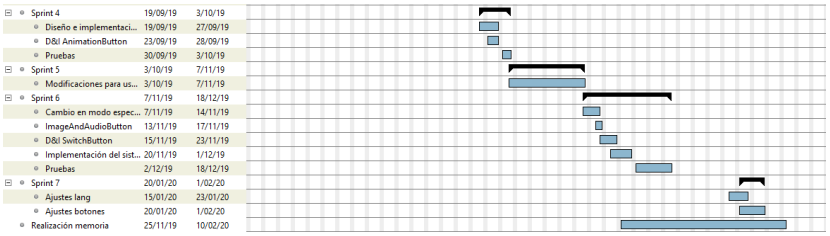


Figura 4.4: Segundo diagrama de Gantt de las tareas del proyecto. Creado con GanttProject

Se puede ver el diagrama completo en el apéndice C.

4.2.1 Sprint 1: Primeros pasos y comprobaciones del modelo

Fecha: 18/06/2019 – 12/07/2019
El primer paso fue realizar una reunión con los dos directores del proyecto para la toma de decisiones iniciales y comenzar así la construcción del TFG. La idea básica era crear un entorno de realidad virtual a partir de los modelos tridimensionales del monasterio de San Julián de Samos creados por la directora Estefanía López Salas[14](Figura 4.5), de modo que se convirtiese en una herramienta de enseñanza científica, cultural y/o educativa.
Por lo tanto, basándose en las funcionalidades extraídas de los dos proyectos mostrados en el apartado Estado del Arte (SpatioScholar y UCLA's reconstruction of World Columbian Expo-

sition), se puso como objetivo construir un mundo virtual que partiese de los modelos del monasterio de Samos y que fuese a la vez entretenido y educativo, para ser usado posteriormente por investigadores especialistas en la materia, museos o asociaciones culturales similares.

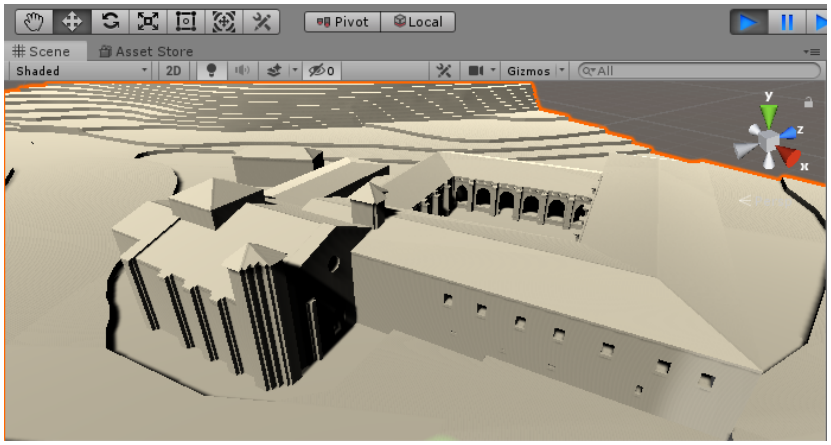


Figura 4.5: Detalle de uno de los modelos visto desde el editor de Unity

Pero previamente a cualquier tipo de diseño o implementación, se debía realizar una comprobación clave: que los modelos disponibles fuesen compatibles con la plataforma escogida para el desarrollo del trabajo, Unity. Se conocía la existencia de proyectos similares que realizaban esa importación a Unity de modelos tridimensionales creados con aplicaciones de Diseño Asistido por Ordenador (los cuales empleaban formatos como *DWG* o *FBX*); pero no estaba claro cómo realizar los pasos.

Puesto que la importación directa del archivo original que contenía la información del modelo (en formato *DWG*) no era compatible con Unity, hubo que buscar la forma de transformar el archivo evitando pérdidas graves de la información asociada a los modelos. Primeramente se intentó usar el software gratuito Blender, con el que ya se disponía de experiencia, pero transformaba los modelos y la información quedaba incompleta en la mayor parte de los casos (Figura 4.6).

Al final, se decidió instalar el software original con el que se crearon los modelos arquitectónicos, Autodesk 3D MAX, y exportarlos a un formato compatible, como es *FBX*. Afortunadamente, Autodesk otorga licencias gratuitas a estudiantes, por lo que tras un proceso sencillo de registro, se logró obtener los diferentes modelos en Unity, e importarlos correctamente.

Una vez que se consiguieron representar todos los modelos disponibles en la plataforma, ya se pudo empezar a diseñar sobre papel la futura aplicación.
Pero previamente a realizar cualquier tipo de diseño, implementación o ajuste, se consideró necesario formarse en el software que se iba a utilizar, Unity.

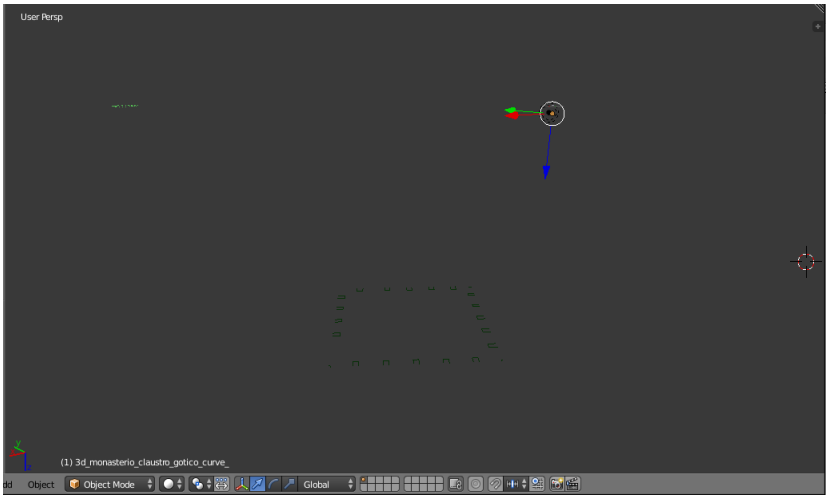


Figura 4.6: Detalle del mismo modelo pero importado desde Blender

Por lo tanto, se dedicaron un par de semanas a acostumbrarse a usar este motor gráfico a través de sus guías oficiales y a los tutoriales que hay disponibles en la web, tanto en formato escrito como en vídeo, realizados por la gran comunidad de usuarios que existe de Unity (Figura 4.7).

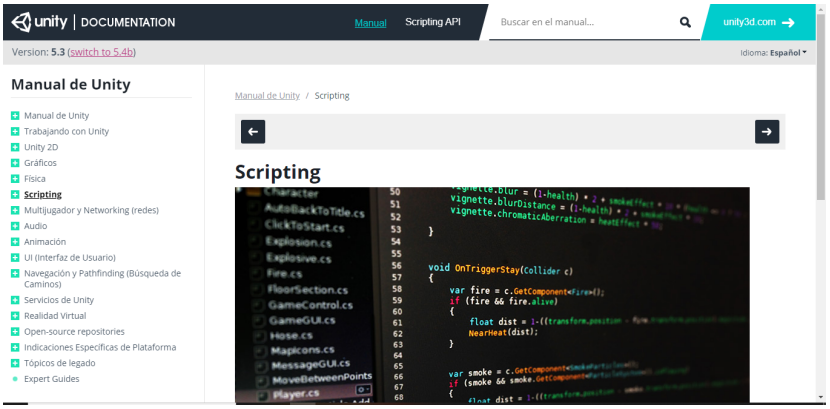


Figura 4.7: Ejemplo de uno de los múltiples tutoriales que ofrece Unity desde su página web oficial

Aunque se había trabajado anteriormente con este software en la carrera (concretamente, en la asignatura de CIIE[1]), apenas fue utilizado y nunca en profundidad, ya que era una más de las tareas que se exigían en la asignatura. Por lo tanto, se partió de un conocimiento previo del entorno casi nulo.

El aprendizaje adquirido de Unity en esta iteración incluye temas básicos como scripts, manejo de la interfaz, materiales, objetos y estructura del proyecto. No fueron más que unos

[1]CIIE: Contornos Inmersivos, Interactivos y de Entretenimiento

preámbulos para poder manejar mínimanente el programa; los conocimientos más avanzados se fueron adquiriendo según se iban implementando los diseños previamente realizados.

4.2.2 Sprint 2: Concretización de las líneas principales de trabajo y diseño inicial

Fecha: 12/07/2019 – 12/08/2019
Al tener relativa libertad a la hora de construir la futura aplicación, dentro siempre de los requisitos e indicaciones dispuestas por los directores, se comenzó diseñando un pequeño prototipo que implementase, de forma muy básica, las principales características que buscábamos:

- Se puede explorar el entorno en primera persona (Figura 4.8).
  Para este paso únicamente se tuvo que importar un prefab de Unity que ya tuviese implementada la cámara en primera persona, y se realizaron algunos ajustes adicionales (altura de la cámara, amplitud, velocidad de movimiento, etc.).

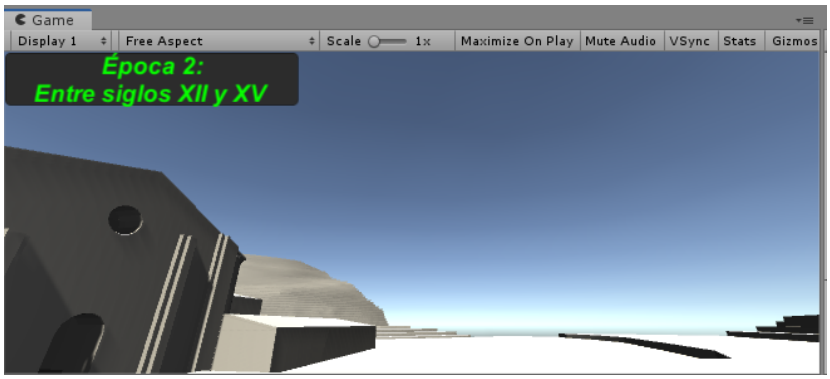


Figura 4.8: Detalle de la vista en primera persona que ofrece el prefab de Unity

- Se pueden ver materiales históricos o contemporáneos usados como base en la investigación (imágenes, fotos, textos, etc.) sobre el modelo o al mismo tiempo que se visualiza. El entorno 3D quedaría muy vacío si tan sólo se mostrase el modelo, por lo que diferentes alternativas que existen para hacerlo más agradable al usuario podrían ser las de UCLA's reconstruction (decorar los modelos con texturas coloridas que lo hagan más atractivo visualmente) o las de SpatioScholar (un menú externo con imágenes o textos que el usuario puede seleccionar con el ratón y abrir y cerrar como ventanas).

  Sin embargo, ninguna de las dos opciones resultaba convincente: la de UCLA entraba en conflicto con lo afirmado en el artículo de Tan B.K. y Rahaman H. sobre el Patrimonio

Virtual[15], y la de SpatioScholar nos parecía poco intuitiva e incómoda, ya que limitaba los movimientos del usuario y el sistema de ventanas era poco agradable.

Por lo tanto, y tomando como inspiración el sistema de observadores del videojuego Horizon Zero Dawn (Figuras 4.9 y 4.10), se diseñó una forma de mostrar texto, audio o imágenes sin limitar demasiado el movimiento del usuario.
A través de un botón en el suelo del entorno, podemos activar un evento que muestre imágenes y/o reproduzca texto, sin impedir al usuario realizar otras acciones. Sin embargo, al alejarse demasiado del botón o al activarlo de nuevo, el evento finalizará. Así, otorgamos la libertad justa para no sentirse forzados a paralizarse y tener que permanecer inmóviles durante la duración del evento a la vez que evitamos que el usuario pierda la atención recorriendo el resto del entorno.

Figura 4.9: El usuario se acerca y activa el evento[3]

Figura 4.10: El evento se reproduce pero el usuario puede moverse libremente[3]

- Se puede cambiar la época histórica mostrada para poder apreciar los cambios en el tiempo del entorno.
  Todos los modelos originales representan el sitio monástico de San Julián de Samos, en la provincia de Lugo. Pero cada uno de ellos recrea el sitio monástico en un período temporal distinto, con cambios que los hacían distinguibles entre ellos. Por ejemplo, si tomamos el modelo del siglo XVII, en él la iglesia sufre una reforma que afecta a una parte de su estructura. Esta reforma está incluida en el modelo y, como consecuencia, éste muestra un cambio espacial con respecto al modelo del período inmediatamente anterior.
  El proyecto SpatioScholar posibilita el cambio de época a través de una barra de desplazamiento, pero no parecía una forma cómoda de realizarlo, ya que se quería centrar el uso del ratón exclusivamente en ajustar la visión del entorno, mejorando así la inmersión de la aplicación (y, de paso, facilitar su uso para entornos virtuales).

La solución a la que se llegó para resolver esto consistió en un submenú donde se escoge la época histórica en la que situarse y, a través de un *HUD*, informar al usuario del período en el que se encuentra actualmente (Figura 4.11).

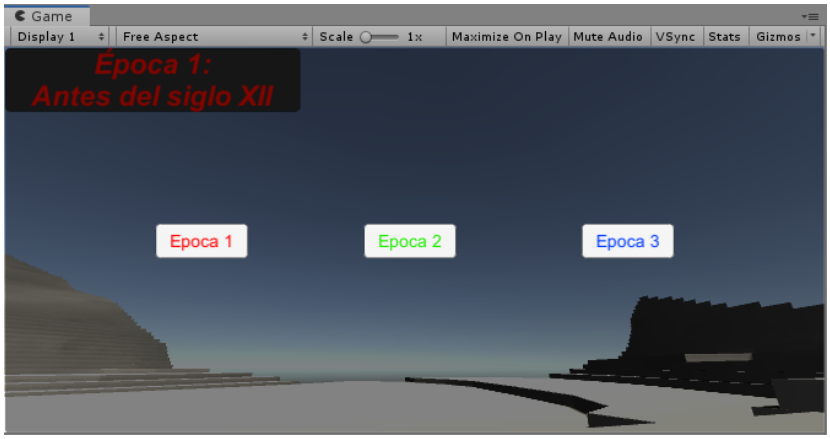


Figura 4.11: Vista del submenú y del indicador de época actual

A mayores, se diseñó el concepto de ”viaje temporal”, un modo especial en el que todos los objetos aparecen independientemente de la época actual, distinguiendo mediante un código de colores la época a la que pertenece cada objeto (Figuras 4.12, 4.13 y 4.14). Por ejemplo, rojo para la época 1, azul para la época 2, etc.

Combinando ambos conceptos, se busca obtener un resultado cómodo para el usuario a la par que atractivo.

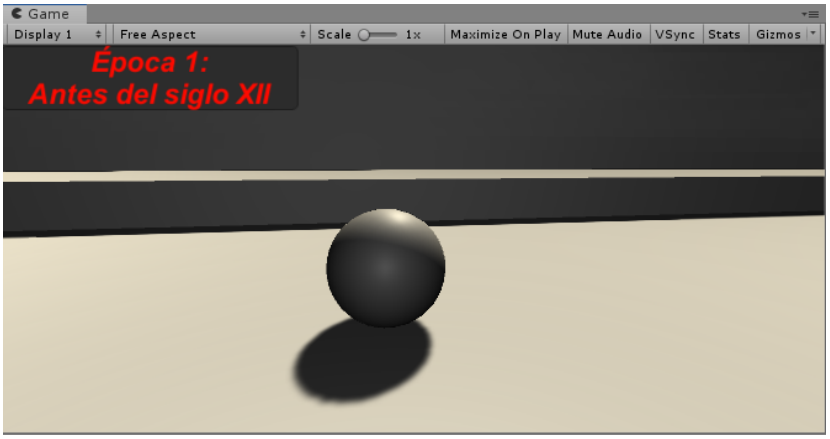


Figura 4.12: Visualización de objetos. Objetos en la época 1.

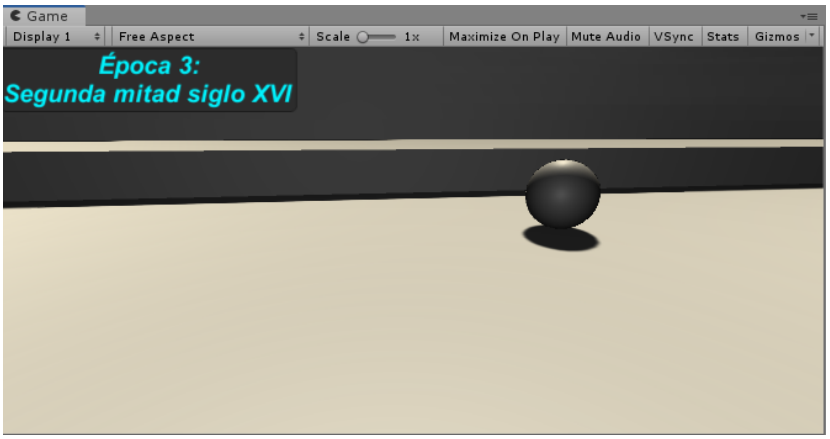


Figura 4.13: Visualización de objetos. Objetos en la época 3

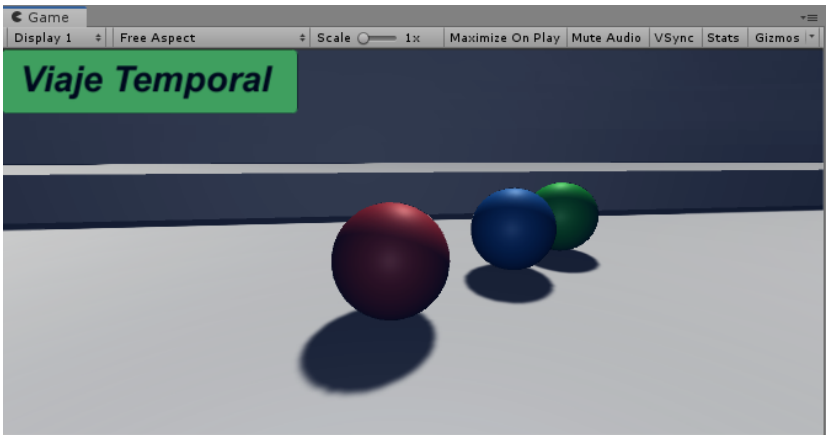


Figura 4.14: Visualización de objetos. Objetos en el modo especial

4.2.3 Sprint 3: Primer enfoque de implementación

Fecha: 12/08/2019 – 19/09/2019
Una vez estuvo listo el diseño sobre papel de un prototipo que cumpliera con las ideas básicas, comenzó la implementación en el entorno de Unity, tomando como base los modelos del monasterio de San Julián de Samos.
Esta plataforma emplea scripts para cada objeto que participe en el entorno, por lo que será ahí donde estarán la mayor parte de las funcionalidades a implementar. Sin embargo, nos encontramos con el problema de que habría que enlazar cada script con sus múltiples parámetros para cada uno de los objetos del modelo, y por este motivo se decidió buscar una posible solución en los foros oficiales de Unity.
Allí se encontró el script "*Mesh Combine Wizard*", creado por Gru[16], acoplable al entorno de Unity y que permite combinar varios objetos en uno solo, evitando así múltiples asignaciones que podrían llegar a resultar tediosas (Figura 4.15).

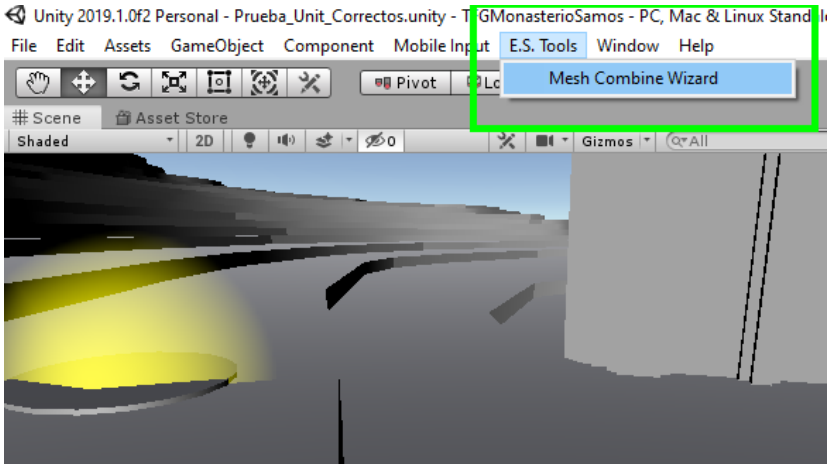


Figura 4.15: Detalle de la interfaz del pequeño script Mesh Combine Wizard

Para el modo especial de la aplicación, se comenzó elaborando el script ModeController, que se encarga de cambiar entre el modo normal y el modo especial.
Paralelamente, se implementó AgeController, el cual se encarga únicamente de abrir un menú en el que podríamos escoger qué época establecer como actual, y AgeDisplay de representar la información en pantalla. Finalmente, se implementó AgeObjectController, que realizará la actualización en cada frame del objeto asociado, dependiendo de si debe estar visible, oculto, o mostrando un material especial (por entrar en el modo especial).
Aunque la actualización de muchos objetos en cada frame suele ser una operación bastante costosa, Unity está especialmente diseñado para trabajar con mucha carga en los métodos de Update. Esta fue la solución encontrada menos costosa a nivel computacional.

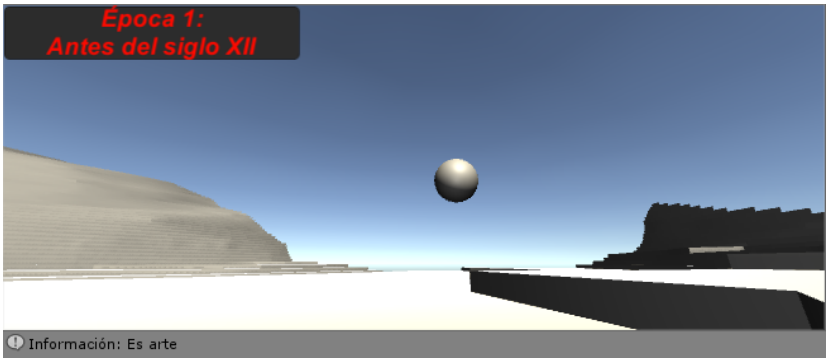


Figura 4.16: Detalle del objeto con el componente InfoButton, mostrando un texto de prueba en el Log de Unity

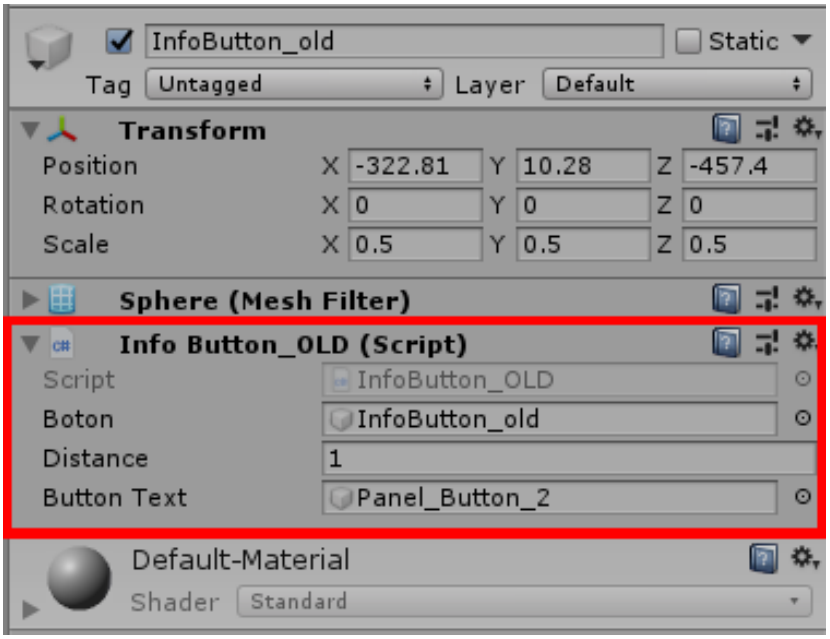


Figura 4.17: Vista desde el editor del objeto con InfoButton

Para una primera versión del botón que reprodujese contenidos multimedia, se creó InfoButton (actualmente sin uso ya que se utiliza para tareas de depuración), la cual simplemente mostraba texto en el Log si estaba a cierta distancia del usuario y se pulsaba con el ratón en dirección al botón (Figuras 4.16 y 4.17).
Posteriormente se desechó el uso del ratón por considerarlo poco intuitivo y se pasó a implementar el ImageButton (renombrado como ImageAndAudioButton), cuyo funcionamiento se activaba al pulsar determinada tecla dentro del rango de activación del botón. Para la prueba, se empleó una instantánea del recinto del monasterio(Figura 4.18).

Para las pruebas, se crearon 3 objetos básicos, cada uno con una época histórica asociada, y

se fue comprobando que cambiaban según la época seleccionada (Figuras 4.12, 4.13 y 4.14).

Figura 4.18: Detalle del botón en el suelo mostrando su imagen asociada

Al final del sprint se realizó una reunión con los directores del proyecto y se mostraron los avances en el desarrollo. Se obtuvo el visto bueno de ambos, pero se recomendó mejorar el cambio de época, ya que el menú (incluído en AgeController) no era demasiado intuitivo y ralentizaba la fluidez del programa al pararlo continuamente.
Además, se decidió aplicar las directrices mencionadas en el libro *"Computers, visualization and history"*[17] en el que se recomienda no emplear texturas para decorar el modelo, ya que aunque se gana en vistosidad y atractivo para el usuario, se pierde fidelidad histórica, algo que debemos evitar en una aplicación orientada al ámbito de la investigación y formación de bienes de interés cultural. Por lo tanto, era preciso buscar otras formas de embellecer el escenario sin reducir el aspecto educativo.

4.2.4 Sprint 4: Cambio de vista, botón animación y etiquetas

Fecha: 19/09/2019 –3/10/2019

A partir de las recomendaciones de los directores en la reunión del sprint anterior, se procedió a añadir nuevos elementos interactivos y a mejorar los ya existentes.

Para otorgar otro punto de vista del entorno y mejorar su visualización global por parte del usuario, se colocó una segunda cámara, otorgando visión del modelo desde una posición aérea (vista cenital), pudiendo apreciar la parte principal del monasterio y la posición actual del usuario. Se implementó el componente ViewController, encargado del cambio entre cámaras y de pausar la ejecución del programa cuando estamos en la vista aérea, y el componente MapIndicator que nos permite distinguir al usuario en esta vista a través de un indicador

(Figura 4.19).

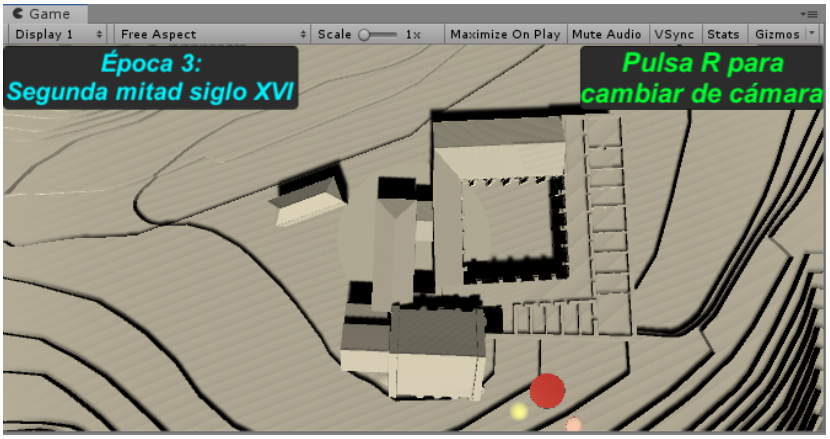


Figura 4.19: Vista aérea del modelo, con el indicador de posición del usuario

Se añadieron otro tipo de botones, con los que se quería aprovechar la separación de elementos producida en la creación de los modelos originales. Por ejemplo, el claustro gótico del monasterio estaba separado en piso, tejado y estructura interna de vigas y columnas. El componente AnimatonButton permite seleccionar varios objetos del entorno e ir distinguiéndolos en un determinado orden, pudiendo apreciar los diversos componentes que conforman lo que el usuario de otra forma vería como un único elemento (Figura 4.20).

Figura 4.20: AnimationButton activado, centrándose en el objeto marcado en color violeta

Hasta ahora se había trabajado con un único modelo del Monasterio de Samos (segunda mitad del siglo XVI), para reducir la complejidad en la implementación de los componentes, y empleando maquetas(*mockups)* para los cambios de época. Se añadieron hasta 3 modelos nuevos del sitio histórico para las épocas: período anterior al siglo XII, entre los siglos II y XV, y principios del Siglo XVII correspondientes a la reforma de la Iglesia.

Aunque el cambio de época funcionaba correctamente, se empezó a distinguir un problema que se agravaría al ir añadiendo más modelos: en el modo especial, aunque aparecen todos los objetos, cada uno lo hace con la textura asociada a su época. Pero, ¿qué sucede con los objetos que aparecen en varias épocas distintas?
Una vez que se añadiesen todos los modelos a la aplicación, existía el riesgo de sobrecargar la carga visual del usuario cuando entrase en el modo especial, al tener que ver demasiados colores a la vez. Como se puede apreciar en la Figura 4.21, el objeto del centro es una mezcla de rojo y verde porque aparece en varias épocas diferentes, lo que puede llevar a confusión al usuario.

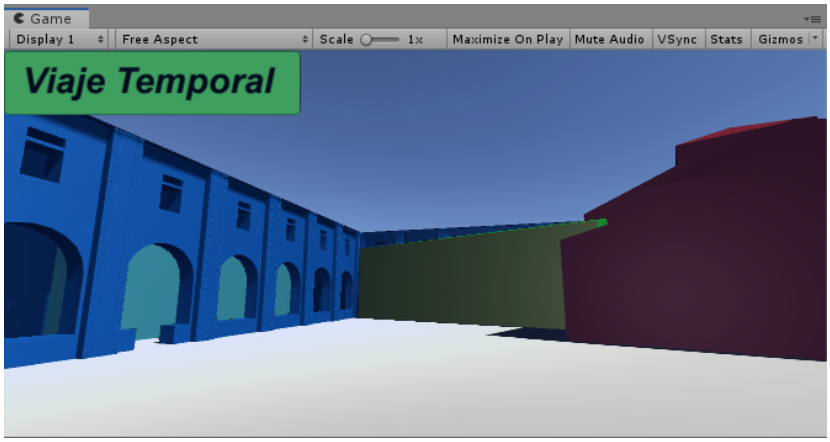


Figura 4.21: Captura de la aplicación en la que se puede apreciar el problema del modo especial.

Entre las posibles soluciones que se plantearon estaba que los objetos fueran cambiando de textura visual cada cierto período de tiempo hasta revelar todos los colores correspondientes a sus épocas, o mostrar cada objeto con una combinación de colores que otorgase información sobre las épocas en las que estuviese activo. Sin embargo, ninguna de esas ideas era suficientemente convincente, pues no arreglaba el problema de la sobrecarga visual, así que se decidió posponerlo hasta la próxima reunión con los directores.

Un importante avance que se logró en este sprint, y que sería vital para el siguiente, fue el conocer el uso de las etiquetas por parte de Unity (Figura 4.22). Gracias a ellas, se evitaría tener que realizar una gran cantidad de asociaciones entre elementos y componentes, al crearlas el programa internamente, y disminuyendo así la carga del desarrollo. Además, se reduce la carga computacional durante la ejecución del programa, al crear las asociaciones casi íntegramente al arranque del sistema. Por lo tanto, a costa de un inicio de aplicación un poco más lento, se mejoró la fluidez y rapidez de la aplicación.

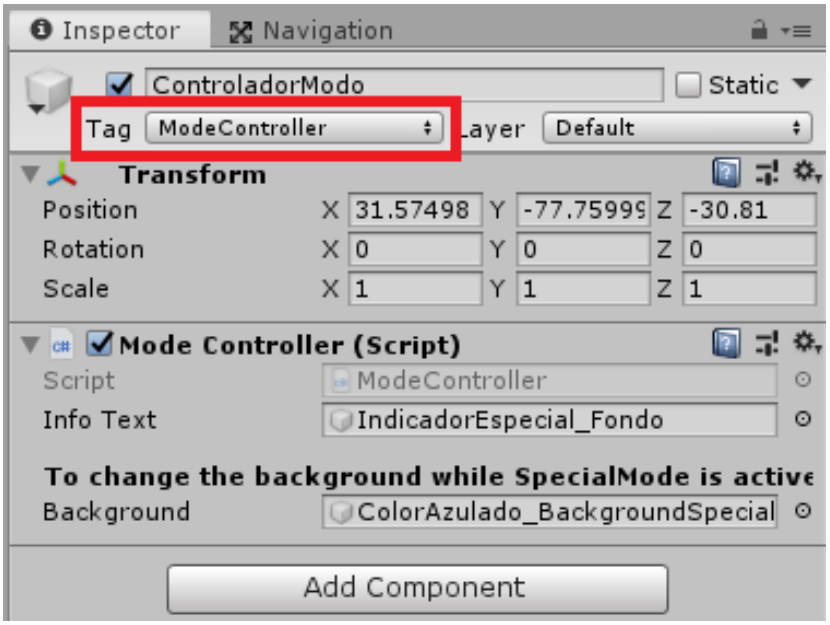


Figura 4.22: Un ejemplo de uso de etiqueta en el componente ModeController

4.2.5 Sprint 5: Creación de Cultunity3D, búsqueda de independencia del modelo

Fecha: 3/10/2019 –7/11/2019
Gracias a las optimizaciones que el uso de etiquetas proporcionó, las interfaces de los componentes quedaron simplificadas y mucho más limpias. Tanto, que se consideró ir un paso más allá de los planteamientos iniciales.

El proyecto estaba orientado a ser una aplicación basada únicamente en los modelos de un único sitio histórico: el monasterio de San Julián de Samos (Figura 4.23). Sin embargo, debido a la forma en la que están diseñados e implementados los componentes, es relativamente sencillo poder independizar las funcionalidades de los modelos. Esto quiere decir que podemos seguir usando todos los componentes creados en este TFG en cualquier otro modelo arquitectónico, sin que su exportación fuera del proyecto suponga un gran coste de esfuerzo o tiempo.

A partir del uso de etiquetas, se inició un proceso de simplificación de interfaces y independizaciones del modelo que llevase a tener una librería de componentes totalmente separada del modelo. El resultado de este proceso es lo que denominamos Cultunity3D, el conjunto de scripts y extras asociados (materiales, imágenes, sonidos) que es totalmente independiente del modelo y simplificado de tal manera que, excepto para cambios muy profundos, un/a posible desarrollador/a puede trabajar e implementar nuevas funcionalidades en su propio modelo sin

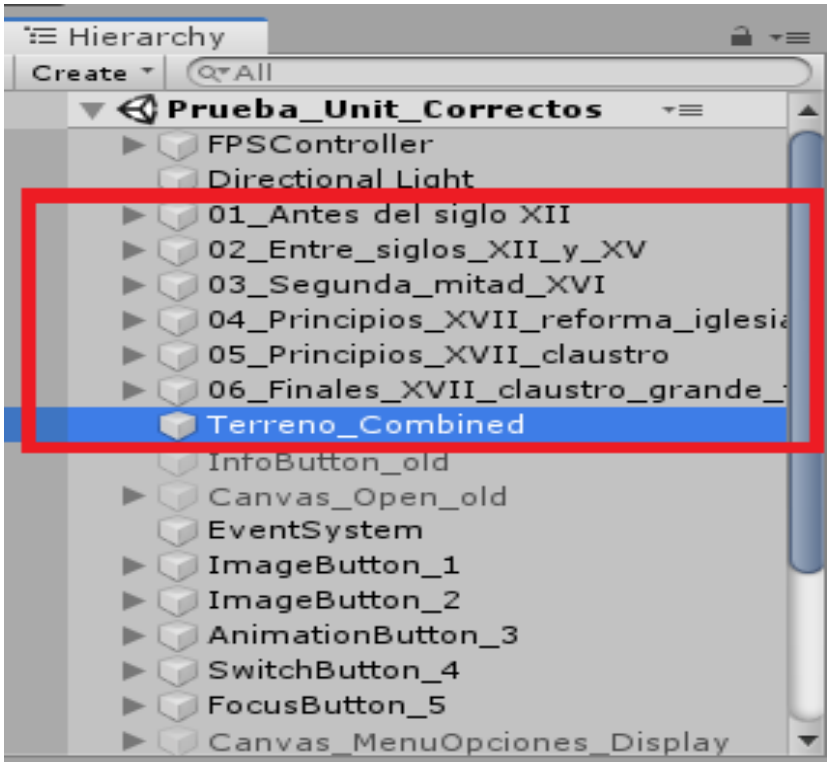


Figura 4.23: Detalle que muestra el conjunto de objetos propios del modelo de Samos

necesidad de tener conocimientos de programación en código C# (Figura 4.24). Únicamente es necesario algo de familiaridad con el entorno de Unity.

Para este propósito, una vez se terminó el último sprint del desarrollo principal del proyecto, se redactó la Guía de desarrollo, que explica todas y cada una de las posibilidades que otorga la herramienta Cultunity3D.

Es importante destacar que, gracias a esta funcionalidad, aunque se cambiase el modelo completamente se podrían seguir implementando botones e interacciones con el usuario sin que fuese necesario introducir ni una línea de código nueva.

Tras crear los nuevos botones y el cambio de vista en el sprint anterior, y de comenzar un proceso de abstracción de los componentes en el sprint actual, se realizó una nueva reunión con los directores, enseñando un segundo prototipo que implementaba los cambios más recientes.

Después de comprobar los avances realizados, dieron el visto bueno, pero consideraron necesario seguir añadiendo diferentes elementos audiovisuales al entorno para atraer la atención del usuario y poder añadir información relevante. A mayores, hicieron énfasis en la necesidad de mejorar la interfaz del usuario y el cambio de época, que seguía teniendo gran margen de

mejora.

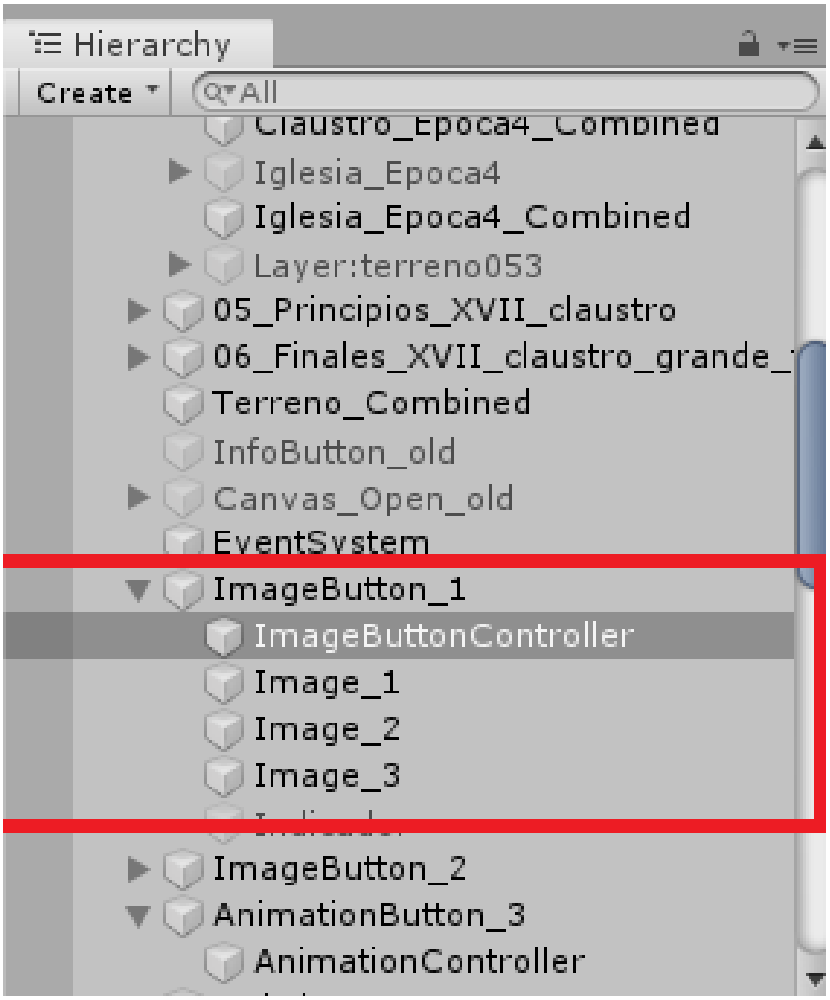


Figura 4.24: Todos los demás objetos del proyecto, como botones o controladores, conforman el módulo de Cultunity3D

4.2.6 Sprint 6: Botón focus, cambio de idioma, y menú de opciones

Fecha: 7/11/2019 –18/12/2019

Tras las conclusiones y acuerdos alcanzados al final del sprint anterior, los objetivos principales pasaron a ser mejorar el cambio de época, arreglar el problema visual del modo especial y añadir nuevos botones para mejorar las funcionalidades.

Puesto que el cambio de época mediante un menú resultaba tedioso y poco cómodo, se decidió eliminar el menú y asignar el cambio de época a unas teclas determinadas, mejorando así la

rapidez de cambio y la fluidez del programa.

Por otra parte, se eliminó la distinción de cada objeto por época del modo especial. En vez de un color para cada época, todos los elementos tendrían la mismo textura, pero ésta sería semitransparente y atravesable. Así, aunque se pierde la información de cada objeto de pertenencia a una cierta época, se gana en fluidez y comodidad visual. Además, la velocidad del usuario aumenta en este modo, lo que lo convierte en una forma ideal de moverse con rapidez por todo el modelo (Figura 4.25).

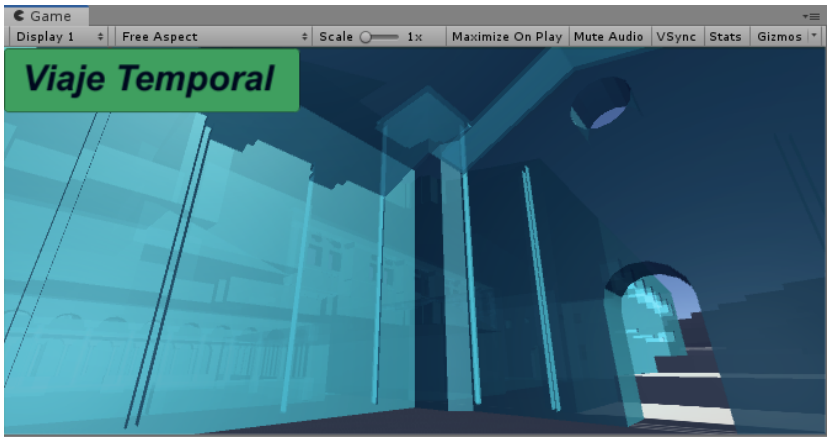


Figura 4.25: Detalle del nuevo y definitivo modo especial

Tras arreglar este problema, se decidieron importar el resto de modelos disponibles del monasterio , lo que no supuso ninguna complicación a mayores, gracias a la forma en que se implementó este tipo de tareas en sprints anteriores.

Se añadieron dos botones nuevos, y se realizaron ajustes en los ya existentes. ImageButton se convirtió en ImageAndAudioButton, permitiendo la ejecución simultánea de audio, para añadir descripciones auditivas al elemento mostrado. También se le incrementó el número de imágenes a mostrar hasta un máximo de cinco. El nuevo botón FocusButton permite seleccionar un único objeto en el que centrarse, pudiendo otorgarle de una descripción tanto en texto como en audio (Figura 4.26).

El botón SwitchButton permite mostrar una imagen pero, a diferencia de ImageButton, ésta no desaparece cuando se abandona la plataforma del objeto, pudiendo mantener la visión de la imagen y apreciar mejor los detalles (Figuras 4.27 y 4.28).

Para el audio y los textos, se añadieron los scripts ChangeAudioLanguage, ChangeTextLanguage y *Lang*; siendo el último creado por Tryder[18]. Permiten el cambio de idioma a partir de un fichero configurable.

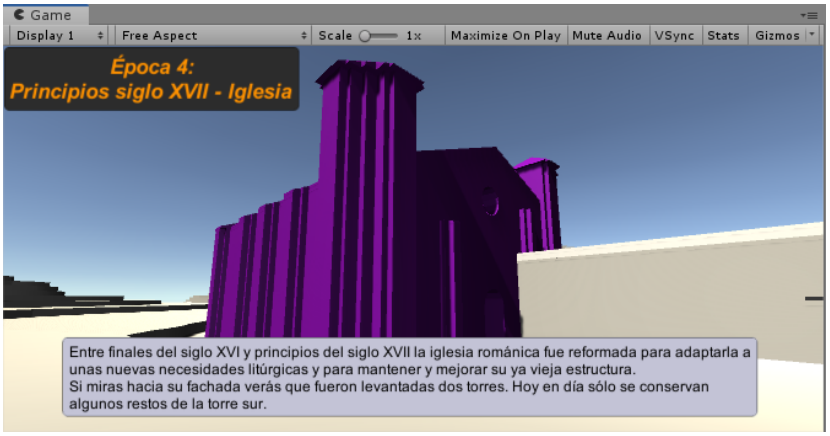


Figura 4.26: FocusButton activado acompañado de audio con subtítulos

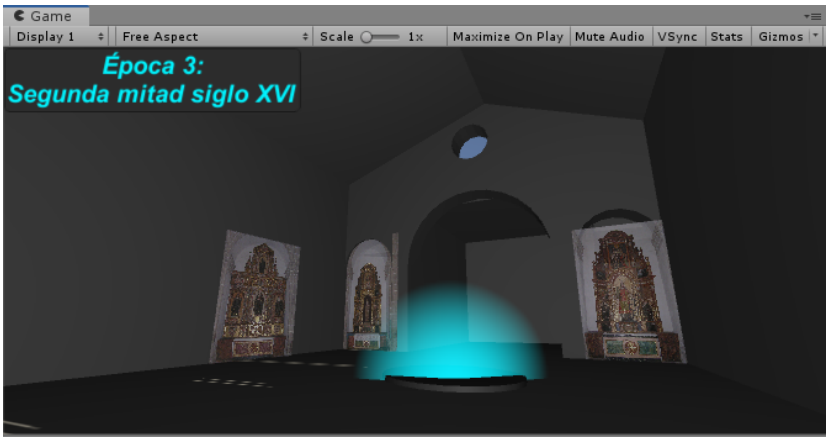


Figura 4.27: SwitchButton activado

Figura 4.28: SwitchButton activado, podemos acercanos a la imagen activada y apreciar sus detalles

Y por último, para realizar diversos ajustes como quitar el sonido de fondo, los subtítulos, cambiar el idioma o pausar la ejecución de la aplicación, se creó el script OptionsController, el cual genera un menú que otorga la posibilidad de realizar los cambios mencionados.

Una vez se completaron todos los objetivos propuestos para este Sprint, se realizó una nueva reunión con los directores para enseñar el último prototipo y comprobar su funcionamiento y el nivel de cumplimiento de los objetivos planteados inicialmente en este TFG.
Una vez revisado y aprobado este prototipo final se consideró terminado el desarrollo de la herramienta Cultunity3D, a excepción de pequeños cambios puntuales.

4.2.7 Sprint 7: Ajustes post-desarrollo principal, y finalización del proyecto

Fecha: 15/01/2020 –01/02/2020
Este último Sprint tenía como único objetivo corregir errores puntuales y preparar el proyecto para ser compilado y ejecutado como una aplicación externa, sin necesidad de abrir la plataforma de Unity.

Fueron necesarios nuevos ajustes al archivo *lang.cs* que contiene el código necesario para ejecutar los cambios de idioma del programa, ya que tal y como estaba anteriormente, funcionaba sólo cuando se ejecutaba desde el Editor. Esto sucedía a causa de la ruta del archivo XML requerido para las traducciones, puesto que se obtenía de manera completa, es decir, la ruta no era relativa a la carpeta Resources.
Una vez que se compila el Build, todos los archivos de Resources aparecerán en un archivo denominado *unity_builtin_extra*[2], dentro de la ruta "Build\TFG Monasterio Samos_Data\Resources". Este archivo es toda la información de la carpeta Resources del proyecto compilada de tal forma que resulta inaccesible desde el exterior.

Sin embargo, desde el archivo *lang.cs* se accedía a una ruta que sólo existía cuando ejecutábamos la aplicación desde el Editor (concretamente, la ruta completa del *lang.xml*). Por lo tanto, al realizar Build, desaparecía la referencia y todos los objetos del entorno asociados a cambios de idioma dejaban de funcionar.

Para arregrarlo se adaptó la lectura del archivo XML para que se realizase desde el módulo Resources, guardando así ese archivo en el posterior *unity_builtin_extra* (y paralelamente, evitando que se pueda acceder al XML desde el exterior del programa). Entre otros ajustes, se modificó la ruta para que Resources la leyera correctamente y se cambió el modo de lectura

[2]Fuente: https://answers.unity.com/questions/381087/everything-on-resources-folder-goes-to-the-build.html

del archivo (en vez de XML directamente, hacer conversión desde TextAsset[3] (ver Figura 4.29).

La detección de los errores se realizó con ayuda del archivo *Player.log*, fichero que está disponible cuando ejecutamos un Build en modo Developer. Así obtenemos una interfaz de Log equivalente a la del editor de Unity.

```
    public void setLanguage ( string path, string language) {
    /* Edit by Adrián Xuíz:
      - Now the lang.xml file must be on Resources, in order to
    work on proyect build.
      - Format of path edited to work with the library Resources.
    */
    path = path.Replace("\\", "/");
    TextAsset textAsset = (TextAsset)Resources.Load(path,
    typeof(TextAsset));
    XmlDocument xml = new XmlDocument();
    xml.LoadXml(textAsset.text);
}
```

Figura 4.29: Parte del código editado para poder cargar correctamente el archivo

Por otro lado, se resolvió un problema detectado en las fases tardías del desarrollo: los botones no se reiniciaban cuando se cambiaba de época mientras éstos estaban activos.

Cuando un botón está activo y se abandona la plataforma desde la que se inicia el proceso, los elementos que participan en la ejecución del botón se desactivan automáticamente (audio sonando, mensaje de *UI* o *CC*...). Sin embargo, si cambiamos la época mientras dura el evento, y en la nueva época no aparece el botón, los elementos quedarían congelados en pantalla y no se podrían eliminar, a menos que se cambiase de nuevo a una época donde sí estuviese disponible el evento.

Aunque parece un problema muy sencillo de resolver, en realidad es un asunto bastante complejo. La desactivación del botón se realizaba en la función OnTriggerExit(), que se ejecuta cuando abandonamos la plataforma del botón. Sin embargo, al cambiar la época, realmente no llegamos a abandonar la plataforma sino que ésta desaparece. Sería necesaria una llamada extra a una nueva función (ejemplo de nombre: ButtonDesactivation), creada con la misión de desactivar manualmente todos los elementos extra del botón.
Pero cuando hacemos desaparecer el botón a causa del cambio de época, lo hacemos desde una función externa, desde la cual sólo podemos acceder a SetActive(). Es decir, no podemos acceder a esa nueva función de ButtonDesactivation().

[3]Fuente: https://answers.unity.com/questions/377514/xml-loading-works-in-pc-but-not-in-android.html?_ga=2.92167450.141248205.1578324679-1163745463.1555433089

Una posible solución sería crear una interfaz Button desde la que sí exista esa llamada, pero por desgracia, aunque en C# se permite la herencia y el uso de interfaces, no funcionan demasiado bien con Unity. Además, todos los scripts que son componentes de objetos ya heredan de la clase MonoBehaviour (Figura 4.30), y puesto que C# no permite multiherencia, esta posibilidad tuvo que ser descartada.

```
1 public class OptionsController : MonoBehaviour
2 public class AgeController : MonoBehaviour
3 public class AgeObjectController : MonoBehaviour
4 public class MapIndicator : MonoBehaviour
5 public class ViewController : MonoBehaviour
6 public class ModeController : MonoBehaviour
7 public class ImageAndAudioButton : MonoBehaviour
8 public class FocusButton : MonoBehaviour
```

Figura 4.30: Código que muestra la herencia de MonoBehaviour de las clases del proyecto

Al final, tras buscar información sobre este problema en los foros especializados, se encontró una solución adecuada: el uso de la función OnDisable()[4]. Ésta es una función que se llama cuando el script se desactiva o se destruye.
Cuando empleamos SetActive() para desactivar un objeto, también se desactivan todos sus componentes, scripts incluidos, por lo que simplemente llamamos a ButtonDesactivation() desde OnDisable (comprobaciones de valores nulos aparte, para evitar fallos de referencia).

A mayores, y para evitar duplicaciones de código, ahora desde OnTriggerExit() también llamamos a ButtonDesactivation().

[4]Fuente: https://forum.unity.com/threads/event-for-when-a-gameobject-is-deactivated-activated.193980/

4.3 Diseño

A partir de los requisitos definidos en la fase de análisis, en cada una de las iteraciones del ciclo del desarrollo el diseño se fue creando previamente a cualquier tipo de implementación. Además, se aplicaron patrones de arquitectura software adecuados a la aplicación y que nos permiten crear una herramienta sin fallos y flexible.

Para los mecanismos centrales, como el control de época, de cambio de modo o de actualización de la interfaz, se ha empleado el patrón **Controlador**[19](Figura 4.31).

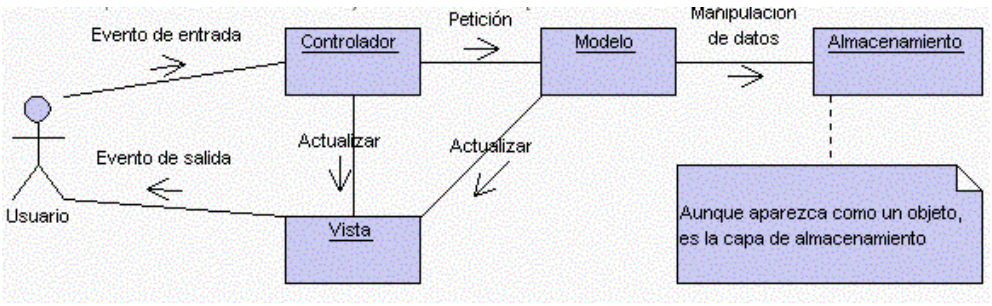


Figura 4.31: Esquema del patrón Controlador

Es un patrón de diseño que nos permite gestionar los cambios del sistema de manera eficiente. Para cada una de las funcionalidades básicas, existe un controlador:

- **Controlador de modo**: Maneja el cambio entre modo normal y modo especial.
- **Controlador de época**: Maneja el cambio de épocas, y la interfaz asociada.
- **Controlador de opciones**: Maneja el menú de opciones y todas sus funciones asociadas (cambio de idioma, salir del programa, desactivar subtítulos, desactivar música).
- **Controlador de objeto**: Se encarga de actualizar cada objeto asociado dependiendo de la época o evento activo. Hay un controlador para cada objeto.

El controlador de objeto se podría haber implementado como un único controlador, pero no sería eficiente ya que necesitaría conocer y actualizar en cada momento todos los objetos del entorno.

Cada uno de estos controladores fueron surgiendo según se necesitaban en las iteraciones, pero siempre siguiendo los principios de este tipo de patrón (Figura 4.32).

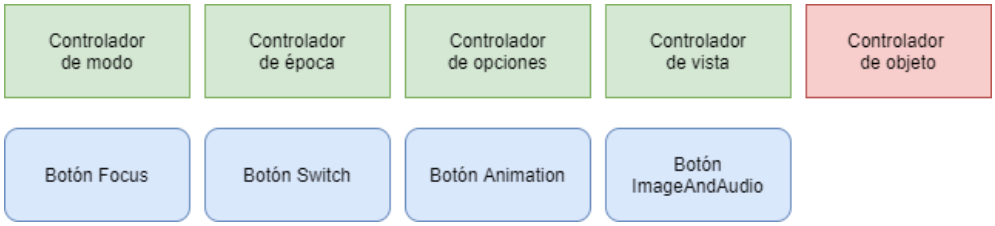


Figura 4.32: Diagrama que muestra los controladores y botones

Para los botones (Figura 4.32), una vez llegaron a un cierto número, se pensó en crear una interfaz común, ahorrando código y evitando duplicaciones. Sin embargo, aunque se realizó su diseño, no se llegó a implementar, debido a problemas con la herencia en Unity: todas las clases heredan de *Monobehaviour*, y no existe herencia múltiple en C#. Por lo tanto, aunque existe cierta duplicidad de código dentro de los botones, las soluciones que otorgan los patrones de diseño no son aplicables en este caso.

4.4 Implementación

Para la implementación se emplearon los elementos básicos de Unity, denominados *Gameobject*. Aunque por si solos no tienen ningún tipo de comportamiento, se les puede otorgar de propiedades ya creadas por la herramienta, como son el *renderizado* de materiales, el contacto con otros Gameobjects u otorgar de cámara al entorno; o crear propiedades personalizadas, como es el caso de los controladores o los botones mencionados previamente en el apartado de Diseño. Estas propiedades se denominan componentes, y cada Gameobject puede tener varias distintas, otorgándoles diferentes funcionalidades (Figura 4.33).

Los modelos importados desde Autodesk 3D Max son leídos por Unity como varios Gameobject enlazados. Para comunicarse con el usuario, se les añadió los componentes *Mesh Collider* y *Mesh Renderer*, y los que debían modificar su aspecto al cambiar de época, se les añadió el componente *AgeObjectController* (Figura 4.34).

Para los controladores, simplemente se les añadió el componente adecuado para cada uno, y se realizaron los cambios necesarios (Figura 4.35).

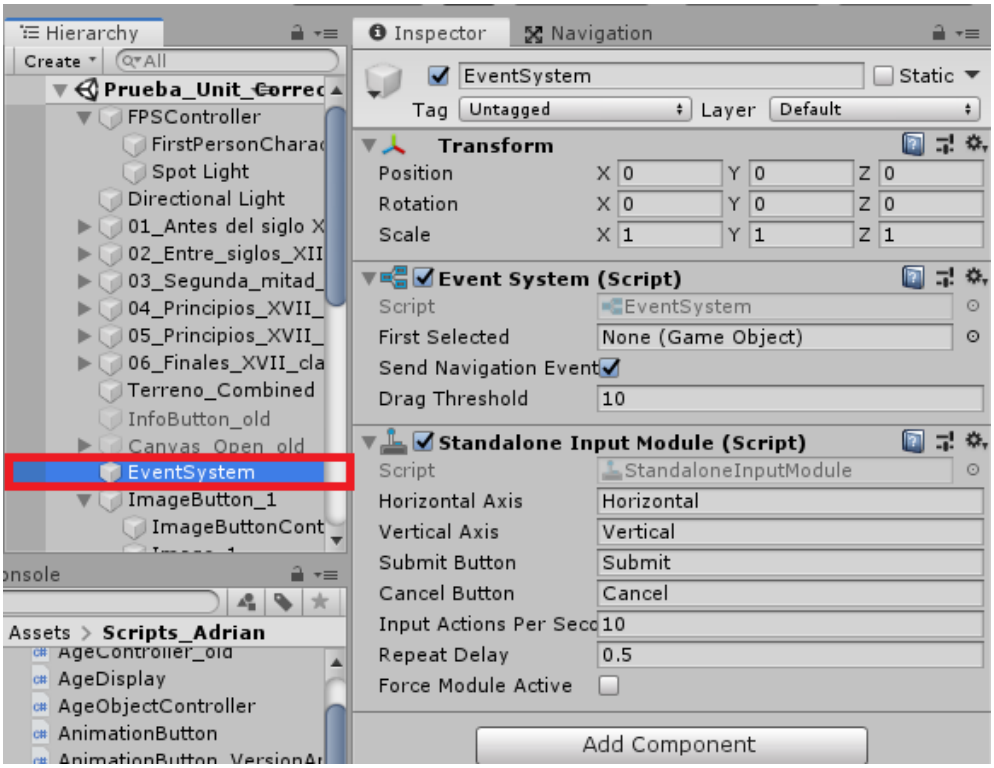


Figura 4.33: Detalle de un Gameobject con diversos componentes a la derecha

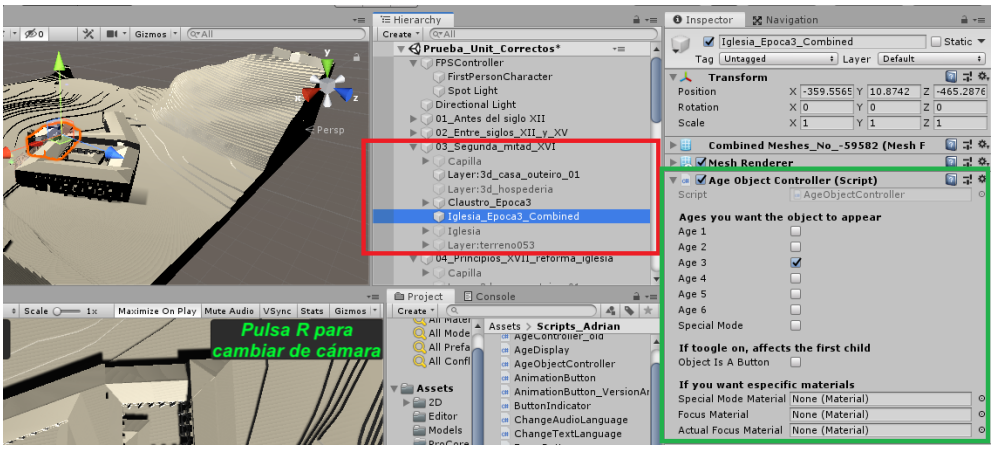


Figura 4.34: Modelo divido en GameObjects, detalle de uno con el componente AgeObject-Controller

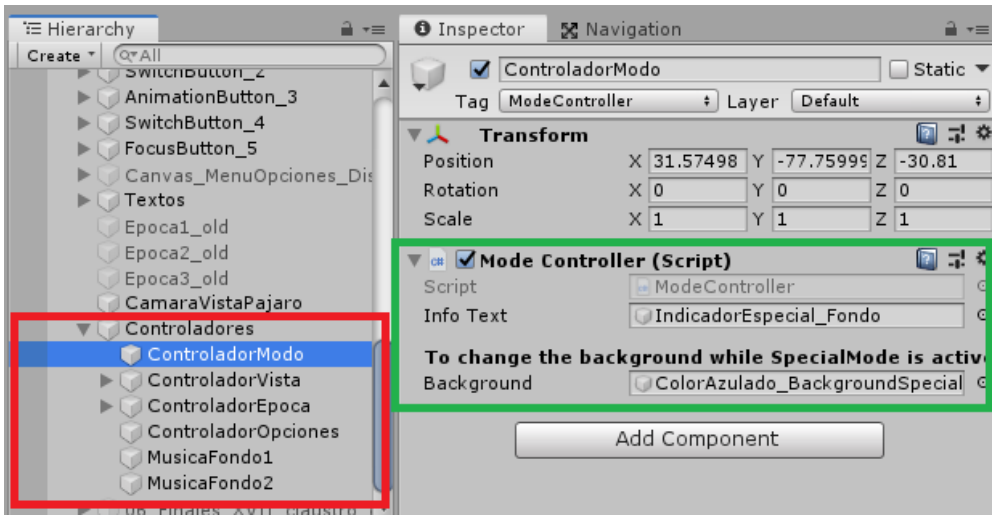


Figura 4.35: Detalle del controlador de modo, con su componente ModeController asociado

En los botones, se les añadió un AgeObjectController, y en los objeto hijo, el componente asociado a cada tipo de botón. De ser necesario a mayores se les incluye materiales, imágenes, hijos extra, etc.(Figura 4.36).

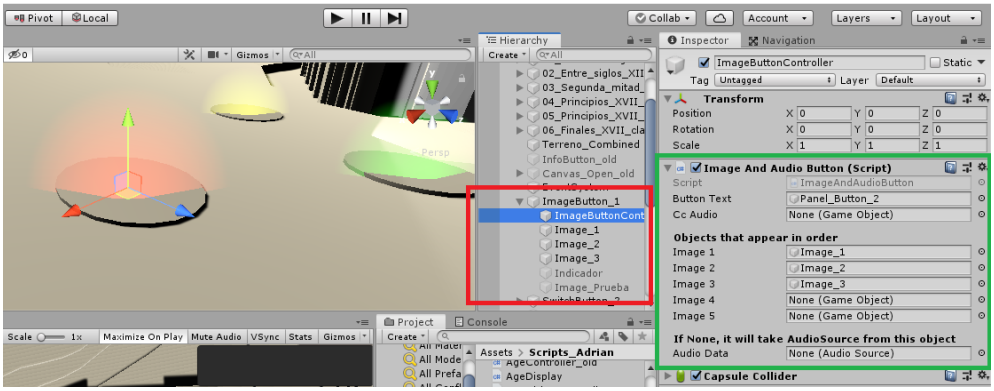


Figura 4.36: Detalle del objeto hijo de ImageButton con el componente ImageAndAudio asociado

Los Gameobject propios de la interfaz de usuario se añaden y modifican con los ajustes necesarios para su correcto funcionamiento. Aquellos con algún tipo de texto tendrán a mayores el componente que les permite cambiar de idioma, *ChangeTextLanguage*, con su referencia al archivo XML (Figuras 4.37 y 4.38).

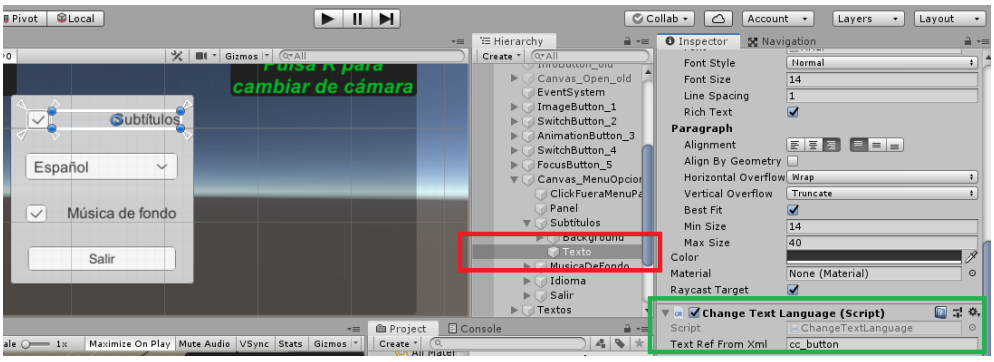


Figura 4.37: Detalle del objeto Texto con el componente ChangeTextLanguage

Por último, se añadió un *Prefab* de primera persona, y se realizaron los ajustes necesarios para adaptarlo al proyecto (Figura 4.39).

```
<string name="age_3">Época 3:
Segunda mitad siglo XVI</string>
<string name="age_4">Época 4:
Principios siglo XVII - Iglesia</string>
<string name="age_5">Época 5:
Principios siglo XVII - Claustro</string>
<string name="age_6">Época 6:
Finales siglo XVII</string>
<string name="cc_button">Subtítulos</string>
<string name="CC_1">Entre finales del siglo XVI y principios del siglo XVII
Si miras hacia su fachada verás que fueron levantadas dos torres. Hoy en día
</Español>


<Galego>
<string name="change_camera">Pulsa R para cambiar de cámara</string>
<string name="image_button">Pulsa E para ver e ocultar a imaxe</string>
<string name="background_music">Música de fondo</string>
<string name="travel_text">Viaxe Temporal</string>
<string name="exit">Sair</string>
```

Figura 4.38: Detalle del archivo XML, donde se definen los textos de los tres idiomas

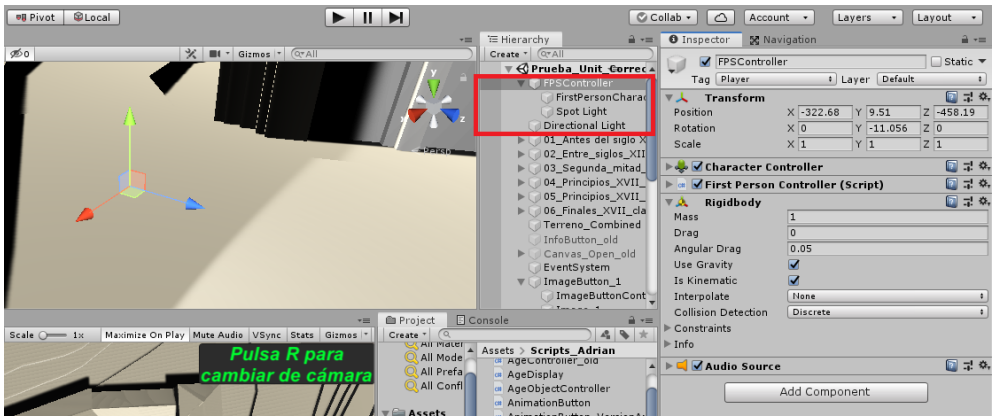


Figura 4.39: Detalle del objecto FPSController, proveniente del prefab de Unity

Capítulo 5

Pruebas

En este capítulo se describen los distintos tipos de validaciones que se realizaron dentro del proyecto. Como se indica en la sección Scrum del capítulo anterior, es necesario realizar un conjunto de pruebas al final de cada iteración para confirmar la correcta implementación de las funcionalidades. Si la aplicación no logra superar las pruebas, es necesario corregir los errores que surjan añadiendo las tareas de corrección a la lista de tareas pendientes, pudiendo provocar retrasos en la planificación del proyecto.

Las pruebas realizadas son las siguientes:

- **Pruebas unitarias**: Validan el correcto funcionamiento de cada funcionalidad implementada.
- **Pruebas de integración**: Validan el correcto funcionamiento de un grupo de funcionalidades.
- **Pruebas de sistema**: Validan el correcto funcionamiento de la aplicación completa como sistema.
- **Pruebas de aceptación y validación**: Validan el correcto funcionamiento de las funcionalidades antes de cada final de sprint, en nuestro proyecto las realizan los directores.

5.1 Pruebas unitarias

En el caso de la herramienta, cada funcionalidad viene implementada por un componente asociado a un *Gameobject* concreto. Se comprueba cada uno por separado, siendo éste el primer

filtro para detectar errores.

5.2 Pruebas de integración

Estas pruebas se realizan una vez se superaron las pruebas unitarias. Los módulos se agrupan en subsistemas y se comprueban en conjunto.
En el caso de la herramienta, se realizaron los siguientes casos de prueba:

- Iteración entre *Gameobjects* de una misma escena.
- Iteración entre los componentes de un *Gameobject*.
- Iteración entre los componentes de distintos *Gameobjects*.

5.3 Pruebas de sistema

Una vez se superan las pruebas de integración, se realizan las pruebas del sistema, en las que se comprueba la funcionalidad global de la herramienta. El objetivo es comprobar que se cumplen todos los requisitos para los que ha sido diseñado.

Para esta aplicación se ejecutaron todos los posibles subsistemas, en distinto orden y comprobando todas las posibilidades.

5.4 Pruebas de aceptación y validación

Tras superar todas las demás pruebas, se ejecutan las pruebas de aceptación y validación, que son las encargadas de comprobar que se cumplen con las especificaciones descritas en el análisis y que los resultados son satisfactorios.

Capítulo 6

Gestión de riesgos

Definimos como riesgo cualquier acontecimiento futuro que pueda afectar negativamente al proyecto. Los riesgos tienen distintas probabilidades de suceder y diferentes impactos en el proyecto, por lo que es necesario estudiar todas las posibilidades y prevenir los riesgos más problemáticos. Se deben calcular los costes de no atajar un riesgo y de resolverlo, para conocer la solución más eficiente[20].

Para la gestión de riesgos en nuestra aplicación se han realizado los siguientes pasos:

- Se ha realizado un estudio de los riesgos en potencia (conocidos y desconocidos).
- Se han estimado las posibilidades de que se conviertan en reales.
- Se ha creado una lista priorizada de los riesgos detectados.
- Se ha realizado un plan de gestión de riesgos, intentando evitar los riesgos o minimizarlos.
- Por último se ha diseñado un plan de contingencia, para el caso de que si finalmente el riesgo se convertía en un problema, el impacto fuese mínimo.

En la Tabla 6.1 se muestran los riesgos identificados para cada uno de los sprints, cuantificando el riesgo en base a su probabilidad de aparición y al impacto que tuviese.

La identificación de los riesgos se ha realizado en base al desconocimiento de Unity y las demás herramientas empleadas, a las capacidades de los recursos materiales y a los compromisos académicos del alumno.

Riesgo detectado	Probabilidad	Impacto	Nivel de riesgo
Falta de conocimiento de las tecnologías	Posible	Menor	Bajo
Diseño inadecuado	Posible	Mayor	Medio
Falta de conocimiento de la metodología	Improbable	Moderado	Medio
Retraso en el tiempo de los recursos	Posible	Mayor	Alto
Pérdida de información en ordenador	Posible	Mayor	Alto

Tabla 6.1: Tabla de riesgos cuantificados

Se resolvieron los riesgos de **Pérdida de información en ordenador**, utilizando un sistema de control de versiones y creando copias de seguridad en un dispositivo externo de almacenamiento; y el de **Retraso en el tiempo de los recursos**, incrementando la duración de cada sprint hasta su máximo (4 semanas).

Capítulo 7

Recursos y costes

En este capítulo se realiza un estudio y estimación de costes de los recursos humanos y materiales empleados para la realización de este proyecto.

7.1 Descripción de recursos

7.1.1 Recursos materiales

Para todos los procesos del proyecto se empleó un único equipo con sistema operativo Windows 10 Home y con las siguientes características:

- Procesador Intel(R) Core(TM) i7-4510U (2.6GHz)
- Memoria RAM 16GB
- Sistema operativo de 64 bits
- Memoria interna de 1 TB
- Tarjeta gráfica AMD Radeon R5 M230

7.1.2 Recursos humanos

Los recursos humanos estimados (pues todos los roles a excepción de los directores fueron asumidos por una única persona) son los siguientes:

- Analista: Es el encargado de analizar y extraer los requisitos del cliente o usuario final, y de establecer los objetivos principales del proyecto.
- Programador: Es el encargado de implementar la aplicación, basándose siempre en el diseño previamente realizado.
- Diseñador software: Es el encargado de realizar un diseño adaptado al problema concreto que la aplicación tiene que resolver.
- Diseñador gráfico: Es el encargado de realizar una interfaz de usuario agradable y cómoda de usar para el usuario final.
- Tester: Es el encargado de diseñar y realizar todas las pruebas que permitan asegurar que el producto final cubre todos los requisitos necesarios.

Dentro de Scrum, todos estos roles se agruparían dentro del equipo de desarrollo.

7.2 Estudio de costes

En este apartado se detallan los costes de los recursos humanos y materiales que fueron empleados en el análisis, diseño, implementación y pruebas de este proyecto.

Para el cálculo del salario de cada uno de los participantes en el desarrollo, se empleará como referencia el *BOE* del *Convenio colectivo del sector de empresas de ingeniería y oficinas de estudios técnicos*[21].

Para este proyecto, suponemos una única persona encargada de los roles de Analista, Programador, Diseñador de Software, Diseñador Gráfico y Tester.

En la Tabla 7.1 se indican los costes estimados de los recursos humanos y en la Tabla 7.2 los costes de los recursos materiales:

Recursos humanos	**Coste/Hora**	**Coste Total**
Analista	40.00€/h	19200.00€
Programador	25.00€/h	12000.00€
Diseñador de Software	30.00€/h	14400.00€
Diseñador Gráfico	20.00€/h	9600.00€
Tester	10.00€/h	4800.00€

Tabla 7.1: Estimación de costes de los recursos humanos

Recursos materiales	Coste Total
Ordenador Portátil Lenovo G50-70	700.00€
Unity (Licencia personal)	0.00€
Notepad++	0.00€
Blender	0.00€
Autodesk 3D MAX (Licencia estudiante)	0.00€

Tabla 7.2: Estimación de costes de los recursos materiales

Los costes humanos estimados se calculan suponiendo 15 horas semanales (es decir, 3 horas diarias durante 5 días a la semana) durante 32 semanas, resultando un total de 480 horas. Aunque no es una distribución eficiente, dado que en una empresa con recursos suficientes se podrían paralelizar tareas y en este proyecto no es posible, consideramos que es una solución factible y realista.

Los costes materiales sólo incluyen el ordenador portátil, pero ese recurso no fue necesario adquirirlo para el desarrollo de esta herramienta, por lo que no se tiene en cuenta en el cómputo final.

Dado que los costes materiales son nulos, el coste total estimado del proyecto asciende a **60000.00€**.

Capítulo 8

Conclusiones

Una vez concluido el desarrollo de este trabajo de fin de grado, podemos considerar que el objetivo principal que se pretendía conseguir se ha alcanzado satisfactoriamente. Así, hemos desarrollado un entorno virtual interactivo e inmersivo, destinado a la representación de bienes de interés cultural, que permite al usuario recorrer y experimentar los cambios espaciales del monasterio de San Julián de Samos en un periodo temporal determinado.

No obstante, también creemos que la eficacia del proyecto aquí presentado requeriría de su puesta en práctica con un mayor número de usuarios. A partir de sus experiencias en el manejo de este entorno virtual podríamos recoger datos sobre muy diversas cuestiones (facilidad de manejo, interactividad, legibilidad, ventajas e inconvenientes del movimiento a voluntad, entre otros) que nos permitirían avanzar hacia su futura mejora. En cualquier caso, en base a las impresiones de los directores, se podría afirmar que supone un avance en cuanto a la situación inicial, en la que teníamos un modelo vacío y sin ningún tipo de interactividad.

Mediante el sistema de botones implementado, el usuario puede comprobar a voluntad aquellas características informativas que desee, y el sistema de navegación por épocas combinado con el modo especial permite un movimiento fluido que invita a investigar el resto del modelo, sin limitaciones pero con pequeños objetivos que ayudan a mantener la atención.

El hecho de implementar la aplicación en Unity permite acceso a un gran número de plataformas, desde PC hasta consolas de videojuegos, pasando por gafas 3D. Aunque sólo exista actualmente el desarrollo para PC, su exportación a otros medios es muy sencilla y no supone apenas costes. En la misma temática, se ha creado una interfaz que facilite el uso de varios idiomas. Todo ello conlleva una mayor accesibilidad a un gran número de posibles usuarios que aporten comentarios y ayuden en futuras mejoras.

Podemos confirmar entonces que, respetando las reglas impuestas para representaciones culturales, hemos aportado una pequeña aplicación que permita el acceso a cuestiones educativas sin perder de vista el objetivo lúdico.

A mayores, la aplicación es casi totalmente desacoplable, lo que permite ser utilizada con otros modelos similares sin suponer grandes costes, y aporta una interfaz de uso casi libre de código, facilitando su implementación incluso para aquellos que no sean expertos en codificación.

En el plano personal, este trabajo me ha servido para trabajar en un proyecto en todas sus fases y comprender cómo emplear los conocimientos adquiridos a lo largo de la carrera de manera eficaz, ayudándome a agilizar y mejorar el desarrollo. Además, he tenido que colaborar con agentes externos a la ingeniería, lo que me ayuda a comprender otros puntos de vista y crear así un software más útil para todos.

Capítulo 9

Líneas futuras

Aunque el proyecto desarrollado cumple ampliamente con los requisitos planteados inicialmente, surgieron diferentes funcionalidades o mejoras que podrían ser implementadas en algún futuro para extender la utilidad de Cultunity3D. Entre ellas, proponemos las siguientes:

- Terminar de implementar la adaptación a un entorno virtual usando gafas 3D, para mejorar la inmersión. El proyecto ya está preparado para ello, pero faltan los recursos materiales que permitan realizar las pruebas y adaptaciones necesarias.
- Ajustar y ampliar la aplicación en cuanto a aspectos tales como menú de inicio, instrucciones o compatibilidad con diversas plataformas, para hacer el producto más interesante y atractivo.
- Realizar ajustes secundarios tales como accesibilidad, mejora de las interfaces, relleno de escenarios, etc.
- Incluir algún tipo de recorrido semi-pasivo que permita recorrer con ciertas limitaciones el entorno, pero que otorgue una experiencia mucho más informativa que el libre recorrido; o emplearse como expositores con recorridos automáticos.

Apéndices

Apéndice A

Guía de usuario

En este apéndice se adjunta la guía de usuario, que explica cómo emplear el sistema desarrollado desde el punto de vista del usuario, y las funcionalidades de las que dispone para moverse por el entorno virtual.

Facultade de Informática
UNIVERSIDADE DA CORUÑA

GUÍA DE USUARIO

Adrián Xuíz García

Versión 1.0

En esta breve guía se describirán todas las posibilidades que ofrece al usuario el proyecto de Cultunity3D.

- Para mover a nuestro avatar, emplearemos las teclas A,W, S y D. Para girar la cámara en cierta dirección, usaremos el ratón.
- Para activar el Modo Especial (Viaje Temporal), pulsaremos la tecla Q. Para volver al modo normal, pulsaremos Q de nuevo.
- Para cambiar de cámara, pulsaremos la tecla R.
- Para activar o interactuar con un objeto del entorno, pulsaremos la tecla E. Para seguir avanzando en las iteraciones, o para salir de la interación (depende del objeto activado), pulsaremos la tecla E de nuevo.
- Para pausar la herramienta, pulsaremos la tecla P. Podemos movernos en el menú y seleccionar opciones con el ratón y el Click Izquierdo. Para volver, pulsaremos P de nuevo.
- Para cerrar la aplicación, pulsaremos F4 o el botón de Salir que aparece en el menú de pausa.
- Si necesitamos ”liberar” el ratón (es decir, que aparezca en pantalla), podremos hacerlo pulsando ESC.

Apéndice B

Guía de desarrollo

En este apéndice se adjunta la guía de desarrollo, que explica los pasos necesarios para utilizar Cultunity3D como desarrollador y todas las posibilidades que ofrece.

Facultade de Informática
UNIVERSIDADE DA CORUÑA

GUÍA DE DESARROLLO

Adrián Xuíz García

Versión 1.2

Índice general

Índice de figuras

Esta guía está orientada para todo aquel que quiera ajustar el proyecto o incluso crear uno nuevo a partir del trabajo ya realizado, a través de la herramienta Cultinity3D. No es necesario ningún conocimiento de C# para realizar estos pasos, pero es conveniente algo de familiaridad con Unity.

Capítulo 1

Primeros pasos: Organización de Cultunity3D

Unity es una plataforma de desarrollo de videojuegos que nos permitirá dotar a nuestros modelos 3D de interacciones personalizadas, como por ejemplo activar una imagen al pulsar un botón o mostrar y ocultar objetos.

Las unidades básicas de Unity son los Objetos, que aparecen representados en la Figura 1.1.

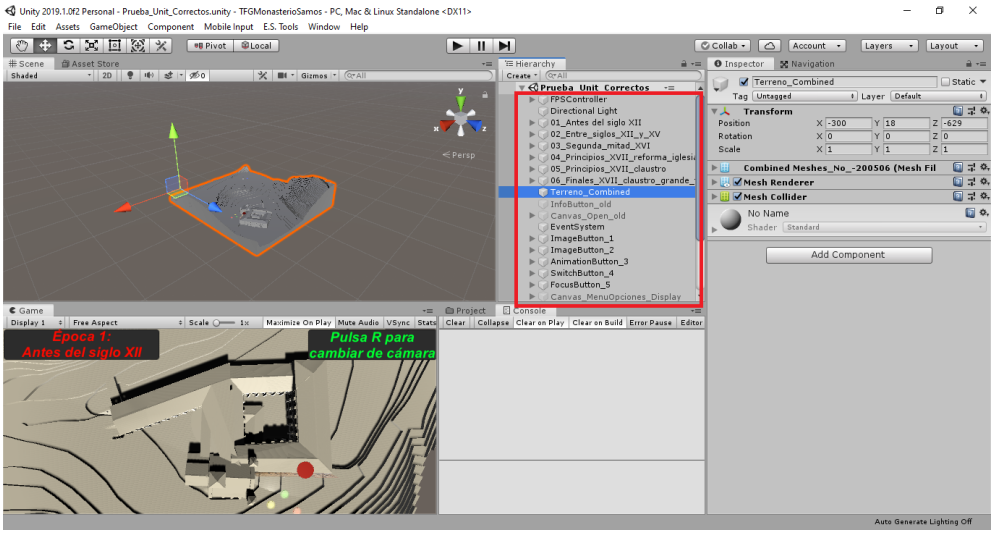


Figura 1.1: Vista de los objetos desde la interfaz de Unity

Cada uno de estos objetos tiene propiedades asociadas, que son los componentes (Figura 1.2). Éstos son los encargados de dotar al objeto de movilidad, posibilidad de iteración, luz, cámaras... Para una comprensión detallada de los elementos clave, se recomienda la lectura de alguna guía básica de Unity.

El trabajo aquí realizado es en su mayor parte componentes específicos, preparados para dotar de comportamientos útiles a nuestro modelo, para representar y explorar el modelo 3D importado. Se recomienda leer la Guía del Usuario para saber qué comportamientos están implementados.

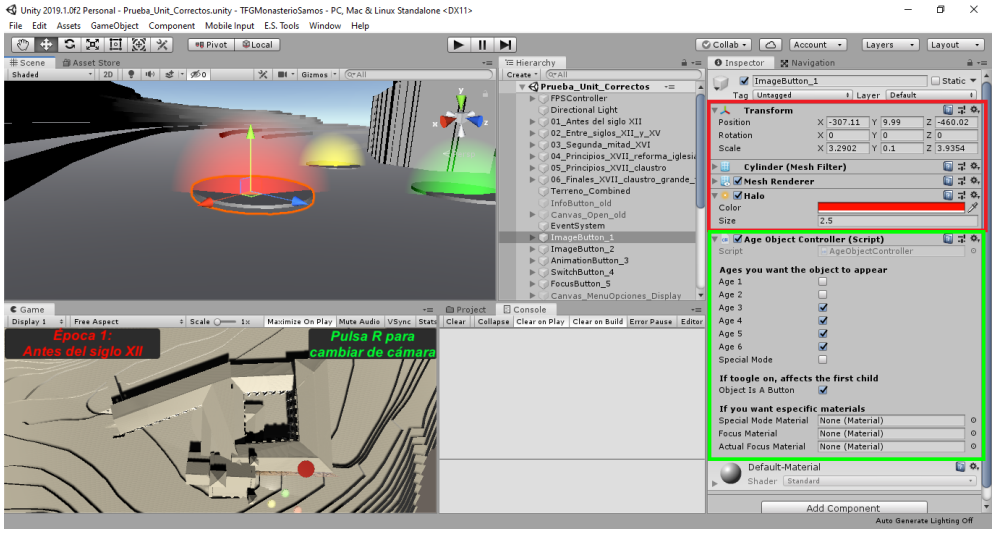


Figura 1.2: Componentes genéricos en rojo, específicos en verde

Los objetos podemos separarlos en 4 grandes grupos: los del propio modelo, los objetos iteractivos, los controladores, y los elementos de interfaz.

Los objetos del propio modelo son los que han sido creado con un software externo (como Blender o Autodesk), es decir, los que queremos dotar de vida propia, pero que por si sólos son unicamente la estructura vacía del entorno (Figura 1.3).

Los objetos iteractivos son aquellos creados con el propósito de dotar al modelo con funcionalidades, como botones en el suelo (Figura 1.4).

Los controladores son objetos vacíos (no se ven) pero que tienen un gran conjunto de tareas asociadas, como controlar el sonido general, los cambios de idioma, el funcionamiento de las cámaras, abrir o cerrar menús (Figura 1.5).

Los elementos de interfaz son esenciales para comunicarse con el jugador. Conforman todos los menús, elementos de HUD, mensajes... Nota: Para verlos desde el editor, activaremos el modo 2D (Figura 1.6).

Aunque se pueden crear cualquiera de estos objetos desde cero, se pueden copiar, pegar y editar los que ya existen en la demo. Los únicos objetos que deben ser nuevos son los objetos del modelo, que conforman la estructura básica de cada nuevo proyecto.

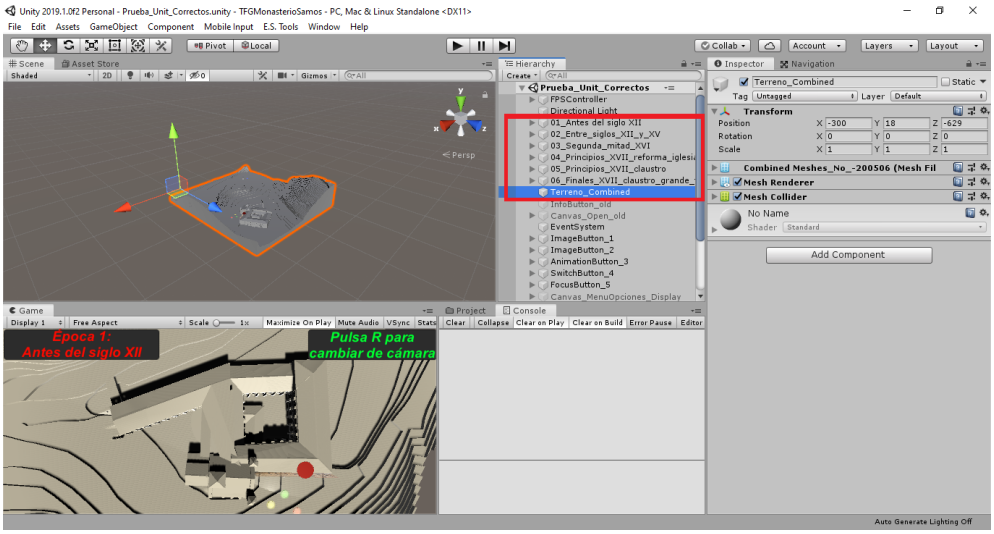


Figura 1.3: Objetos del modelo original

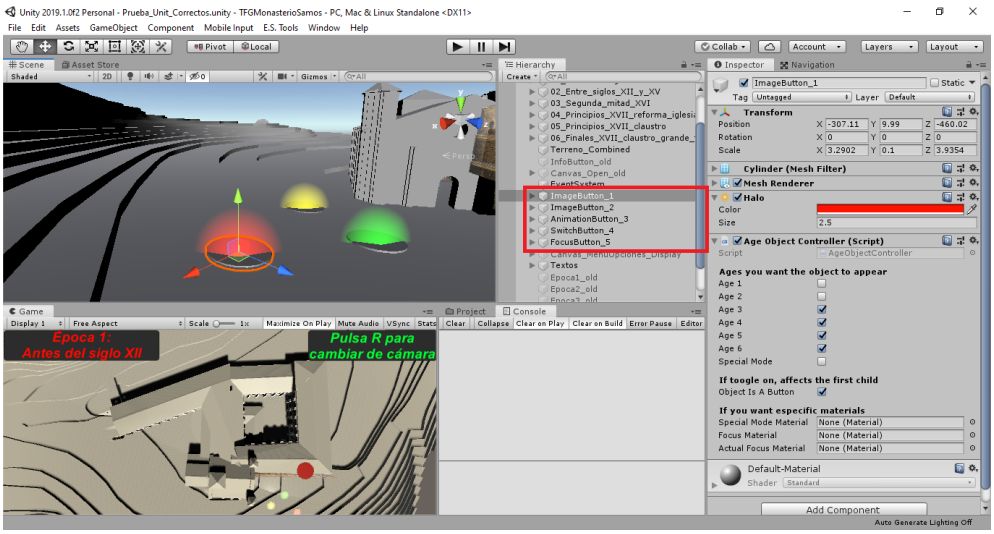


Figura 1.4: Objetos iteractivos

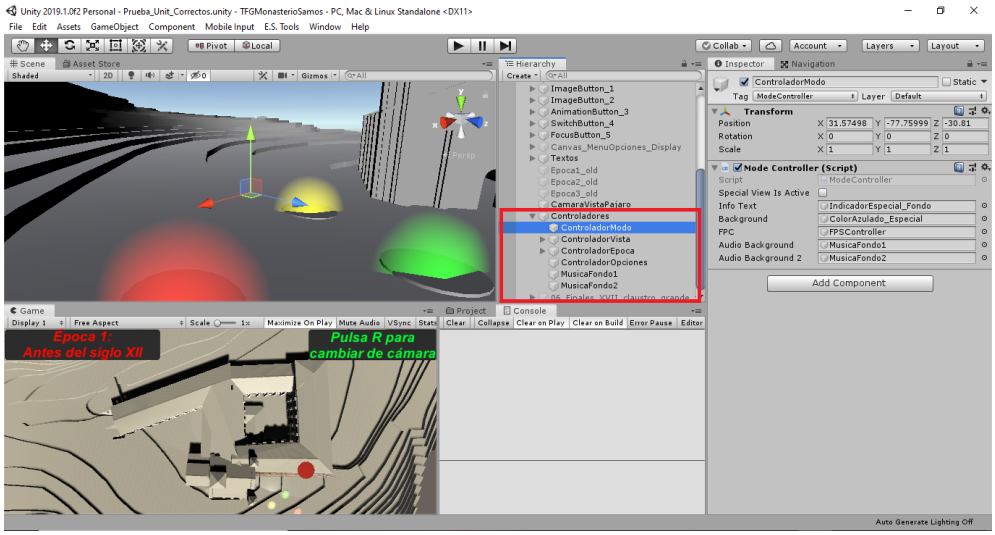


Figura 1.5: Objetos controladores

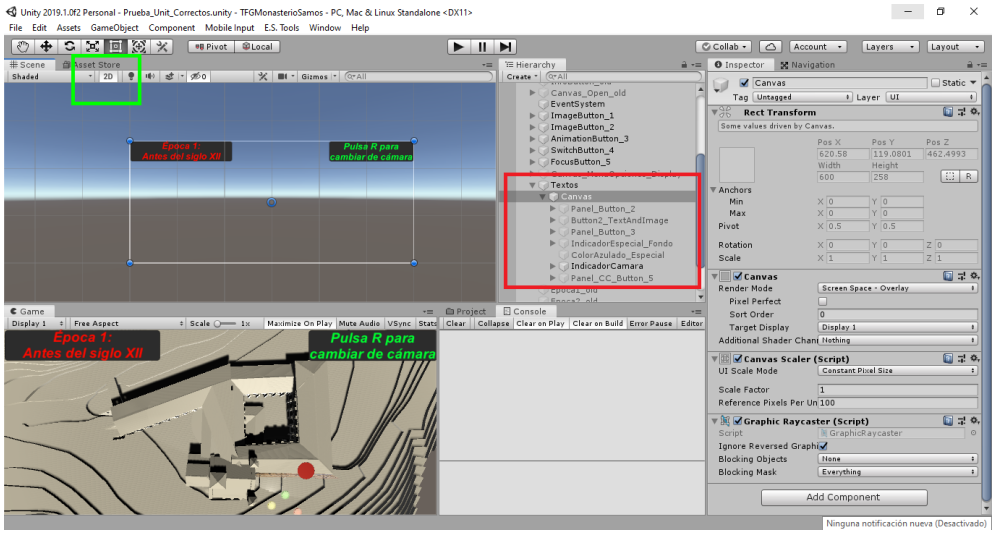


Figura 1.6: Objetos de interfaz

Capítulo 2

Empezando: importación del modelo

Para tener algo minimamente funcional, debemos importar los objetos del modelo en un formato adecuado para Unity. En la demo se empleó el formato *.FBX* (los archivos estaban originalmente creados en dwg pero se exportaron a FBX desde Autocad 3D max).

Una vez lo hayamos realizado, deberíamos tener algo como lo que aparece en la Figura 2.1.

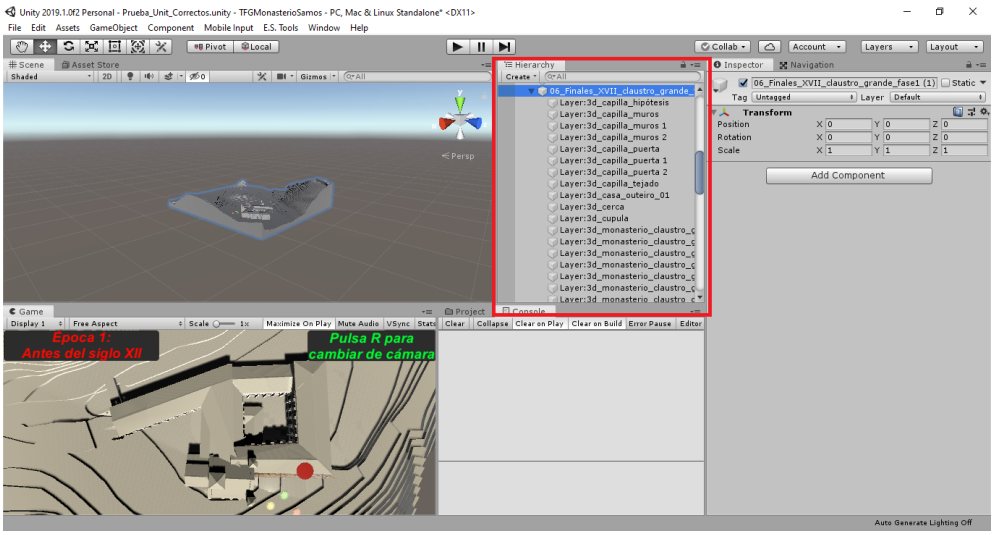


Figura 2.1: Modelo importado desde archivo FBX

Ahora podemos organizar las capas según nos parezca adecuado mediante relaciones padre-hijo. Por ejemplo, podemos crear un objeto vacío llamado ”terreno” y agrupar todas las capas del terreno bajo ese único objeto (Figura 2.2).

El siguiente paso debería ser añadir un prefab de cámara en 1º o 3º persona. Este objeto nos permitirá controlar la cámara y movernos por nuestro entorno recién añadido.
Podemos buscarlo entre los *assets* gratuitos en la Unity Store, o copiar y pegar el que ya tenemos en la demo, el cual es en 1º persona (Figura 2.3).

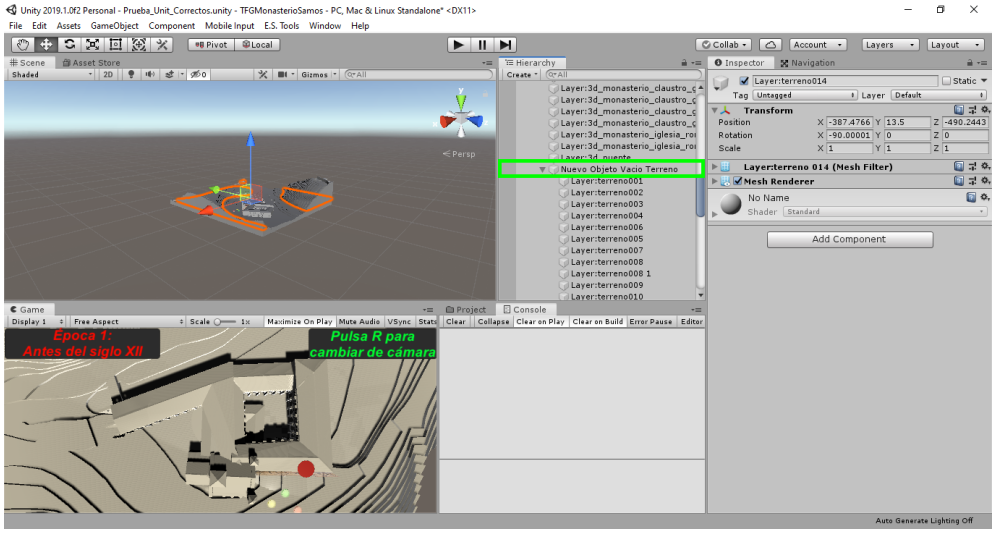


Figura 2.2: Nuevo objeto que actúa como padre

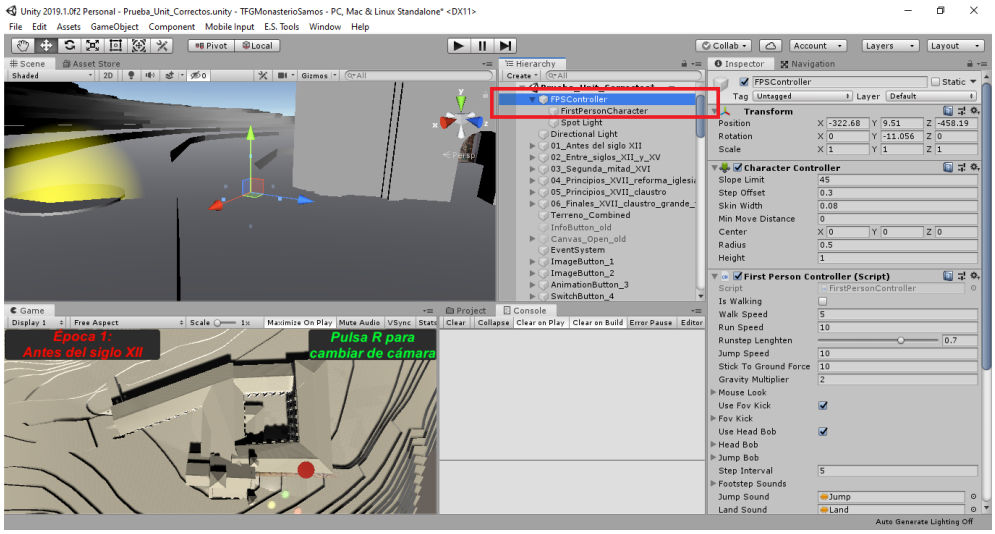


Figura 2.3: Objeto FPS de la demo

Sin embargo, todavía no podemos darle al Play y explorar nuestro modelo. Hay que otorgar a ciertos elementos de la escena de un componente básico: el *Mesh Collider*. Este componente de Unity es el que permite que el prefab que controlamos de 1º o 3º persona (a partir de ahora, Jugador) no atraviese el objeto y pueda chocar contra él. Sin él, nuestro Jugador no sería más que un fantasma que atraviesa todos los objetos, incluyendo el suelo.
Por tanto, a TODOS los elementos contra los que queremos que el jugador pueda colisionar, debemos añadirle un Mesh Collider (Figura 2.4).

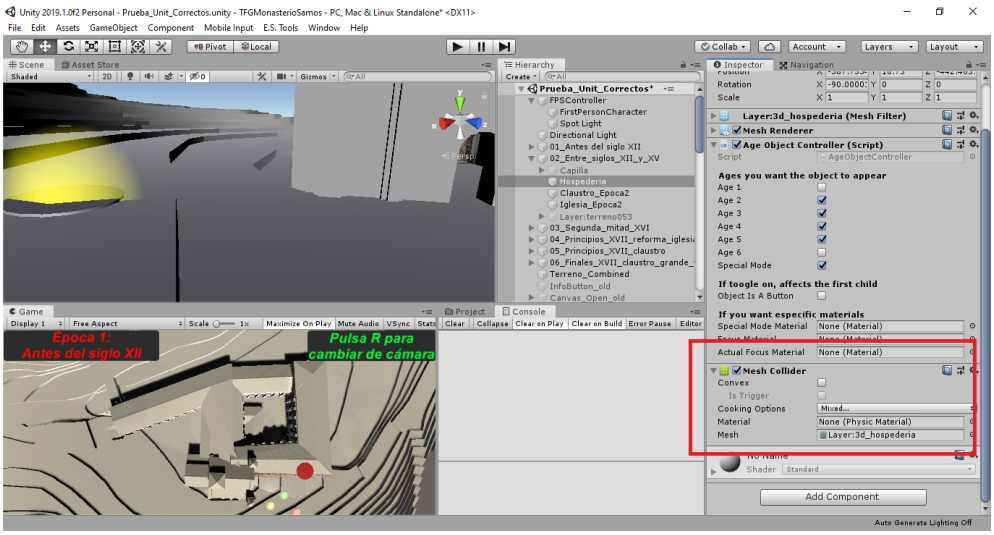


Figura 2.4: Componente Mesh Collider

Pero, ¿qué pasa si tenemos muchos objetos de terreno, como es el caso de la demo? ¿Debemos añadir un Mesh Collider a cada uno de ellos? La respuesta es sí, pero por suerte, en la demo viene incluída una pequeña aplicación que nos permite agrupar varios objetos en uno sólo. De esta manera, sólo habría que añadir un único Mesh Collider a un único objeto. Para ello, seleccionamos en esta ventana el *Mesh Combine Wizard* y ponemos el padre de los objetos que queremos agrupar. En nuestro caso, el objeto vacío que creamos para agrupar el terreno (Figuras 2.5 y 2.6).

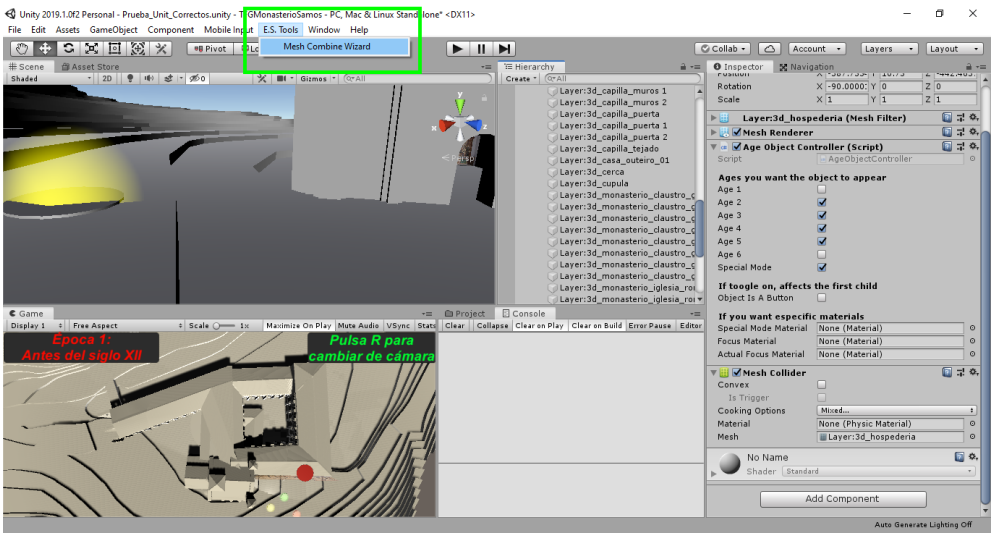


Figura 2.5: Mesh Combine Wizard desde el editor de Unity

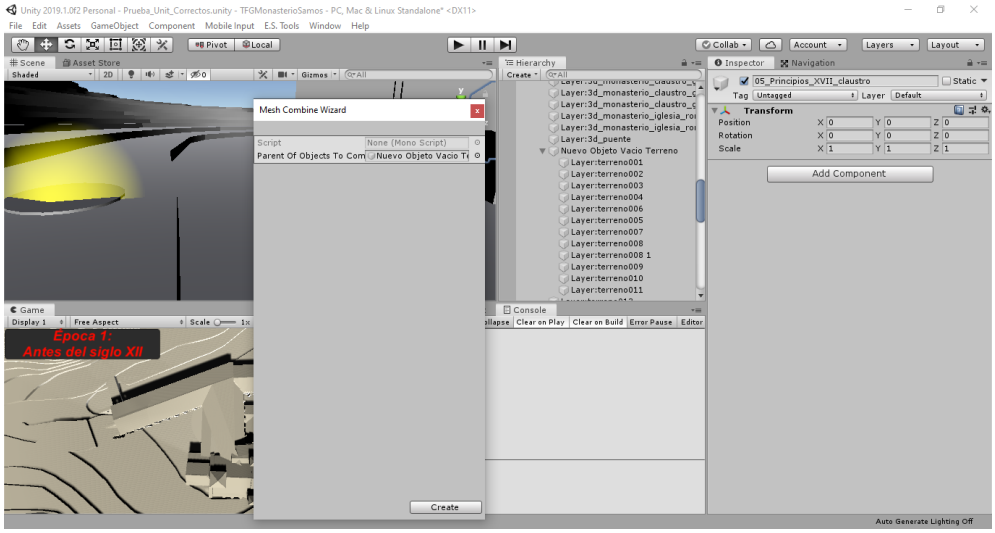


Figura 2.6: Usando el Mesh Combine Wizard

Ojo cuando agrupemos muchos objetos distintos, ya que puede dar problemas si son muy distintos o están en posiciones muy alejadas entre sí. Es conveniente revisar el nuevo objeto creado y comprobar que todo esté correcto.

Una vez le demos al botón de create, el objeto padre que hemos seleccionado y sus hijos se desactivarán, y un nuevo prefab con los elementos unidos aparecerá en el proyecto (Figura 2.7).

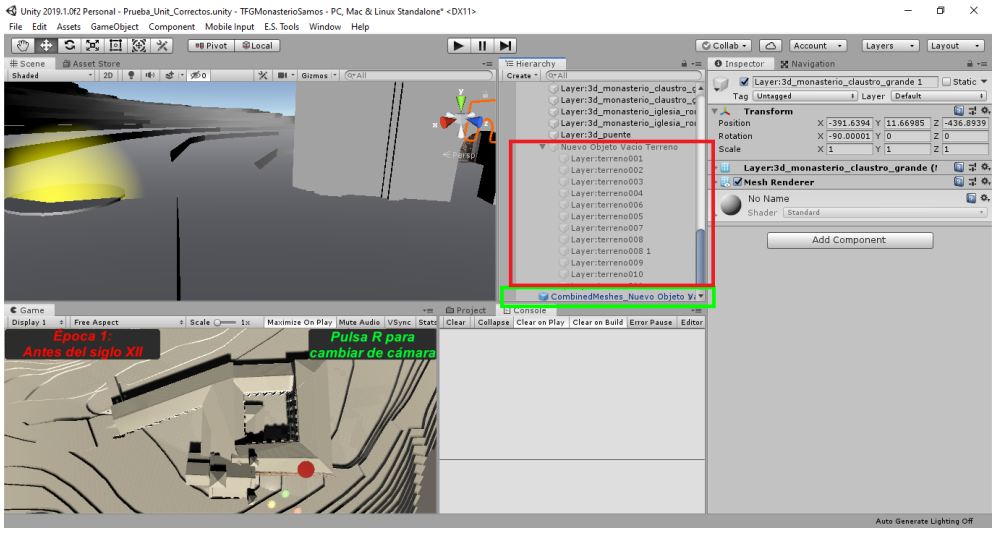


Figura 2.7: Objetos desactivados en rojo, nuevo prefab en verde

Sobre el nuevo objeto, click derecho y seleccionaremos **Unpack Prefab**. Así, el objeto pasará de color azul a blanco (en la vista del editor) y podremos editarlo a nuestro antojo.

En esta imagen podemos ver que el antiguo objeto vacío, que servía como padre de los terrenos, está desactivado (podemos activarlo y desactivarlo pulsando en el *box* del editor, ver Figura 2.8). Nota: aunque los hijos estén activos, si su padre está desactivado, no contarán como activos.

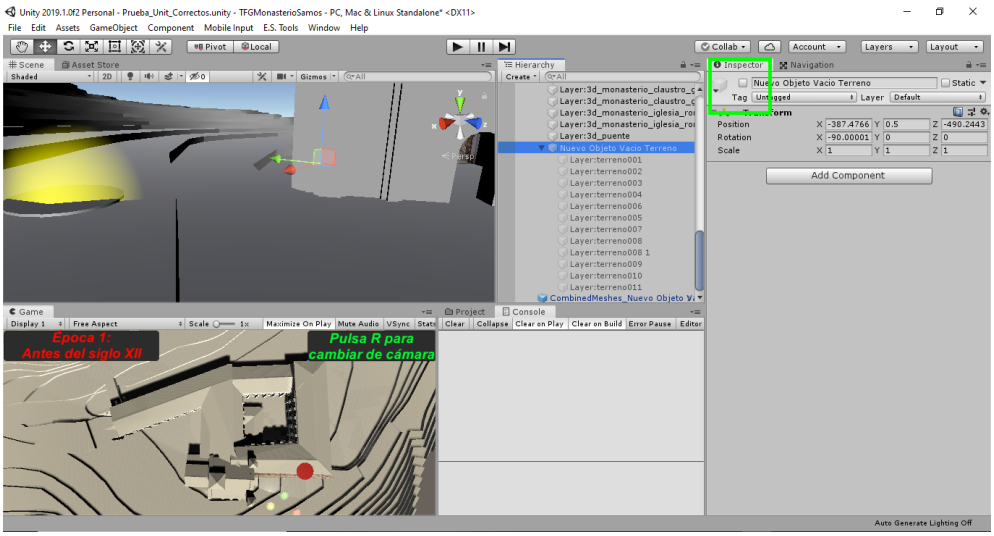


Figura 2.8: Casilla donde activar y desactivar objetos

Una vez hemos desempaquetado el prefab, vamos a añadirle el componente Mesh Collider. Para ello, pulsamos sobre **Add Component** y buscamos *Mesh Collider*. Pulsamos encima y listo, ya tenemos nuestro terreno preparado para ser explorado por nuestro jugador (Figura 2.9).

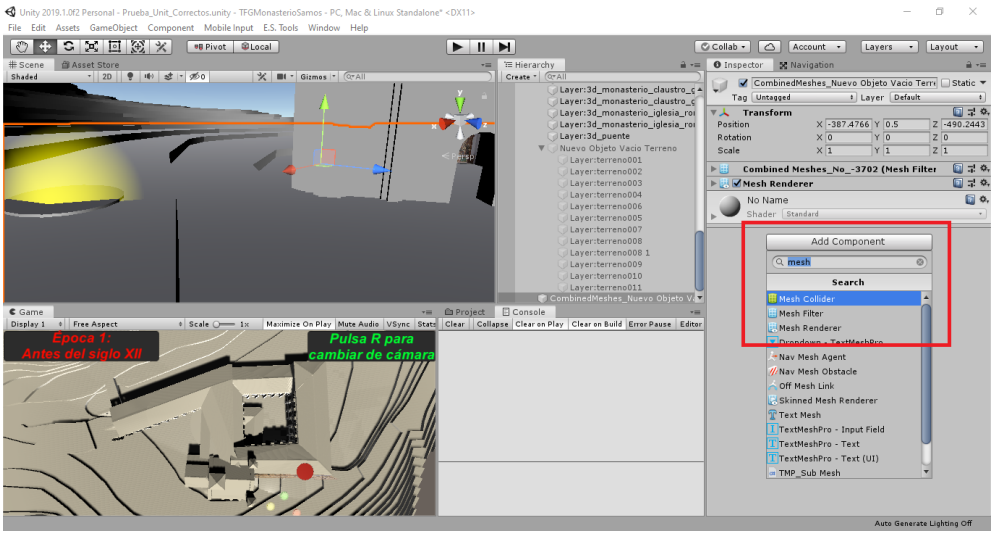


Figura 2.9: Buscando el componente Mesh Collider

Si ahora pulsamos Play, nuestro Jugador debería de poder moverse por el terreno y atravesar el resto del modelo que hemos creado (para evitar que atraviese los objetos, repetimos los mismos pasos que con el terreno). Nota: Si todo se ve negro al darle al play, tal vez sea necesario añadir algún tipo de objeto Luz al entorno.

Capítulo 3

Características especiales de la demo

3.1 Etiquetas

Una de las características especiales de esta demo que hay que tener en cuenta es su uso de las Etiquetas. Las etiquetas son utilizadas por Unity para identificar objetos de manera precisa, y hay que tener en cuenta que son imprescindibles para el correcto funcionamiento de los scripts específicos de la demo. En la demo ya están todas las etiquetas asignadas a los objetos necesarios, pero es algo que debemos hacer manualmente si creamos el proyecto u objetos como los controladores de 0.

El Controlador de Modo tiene la etiqueta ModeController (Figura 3.1).

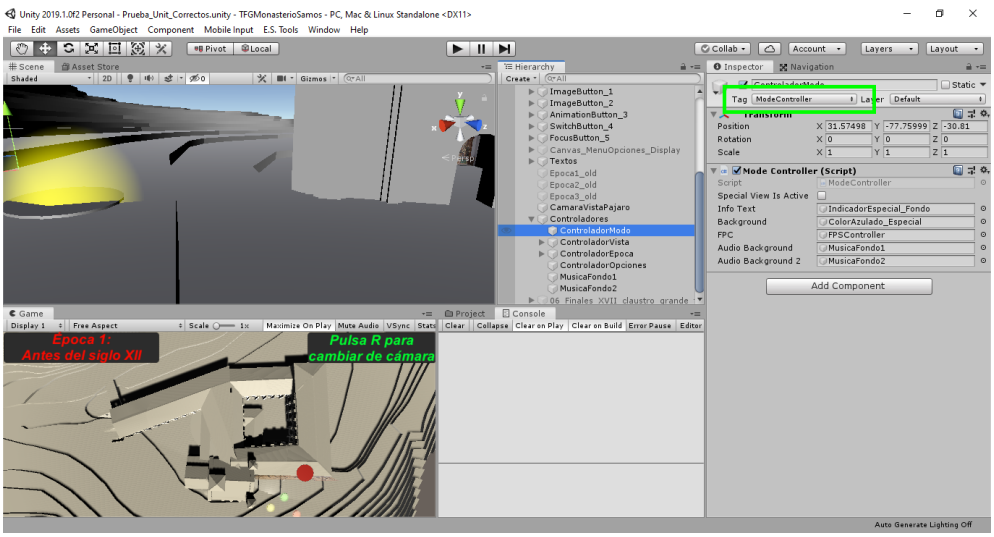


Figura 3.1: Tag/Etiqueta en el objeto

Por defecto, los objetos no tienen etiqueta. De ser necesario, habría que crear la etiqueta específica para cada caso y asignarla al objeto (en la explicación de cada script viene indicado

si hace falta etiqueta).

3.2 Épocas y modo especial

En la demo se muestran 6 épocas distintas, cada una con su modelo correspondiente, entre las cuales podemos ir cambiando. A mayores, si pulsamos determinada tecla, accedemos al Viaje Temporal (SpecialMode) en el que corremos más rápido, atravesamos las estructuras, y podemos ver a la vez casi todos los elementos de cada época.

3.3 Materiales

Los materiales son los elementos del proyecto que dan color a los objetos. Se pueden cambiar los que aparecen en la demo, pero los nombres deben ser los mismos ya que esos materiales son los tomados por defecto por los scripts.
En concreto: Focus, ActualFocus, Default y SpecialMode.

3.4 Scripts/componentes específicos

Estos son todos los scripts realizados por *Adrián Xuíz García* y que vienen incluidos en la demo. Si no se dice lo contrario, basta con añadir el componente a un objeto para que el script funcione. Se recomienda comprobar la demo para ver cómo sería una implementación correcta. Podemos distinguir 4 tipos de scripts según su funcionalidad: de interacción, de controlador, de interfaz y otros. Suele corresponderse con el tipo de objeto al que van asociados. Los scripts de controlador e interfaz deben estar presentes OBLIGATORIAMENTE, ya que contienen las funcionalidades básica del proyecto (apertura de menús, cambio de época, actualización del hud...) NOTA: Todos los elementos con etiqueta tienen un script específico asociado a EXCEPCIÓN del prefab del Jugador, que deberá tener la etiqueta Player, y el foco de luz asociado a este, que deberá tener etiqueta PlayerLight. En la demo, este será el objeto FPS.

3.4.1 AgeController

Controlador
Script que controla el cambio de época. El objeto que tenga este componente deberá tener la etiqueta AgeController. Para su correcto funcionamiento, las etiquetas ModeController, AgeDisplayController y Player deben estar asignadas (Figura 3.2).

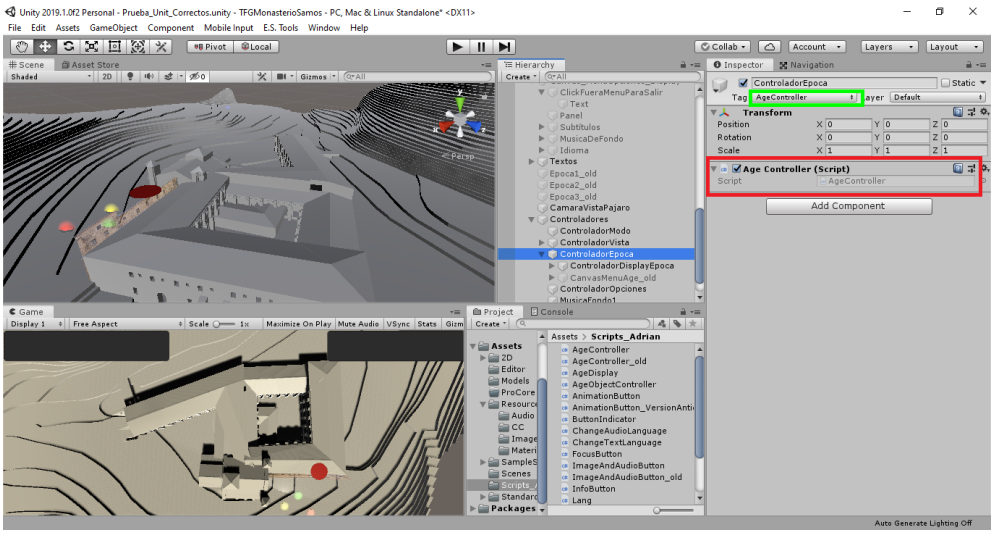


Figura 3.2: AgeController y su tag

3.4.2 AgeDisplayController

Interfaz

Script que controla la interfaz de usuario que muestra la época actual. Debemos enlazar cada uno de los objetos de Text que muestra una época. El objeto que tenga este componente deberá tener la etiqueta AgeDisplayController (Figura 3.3).

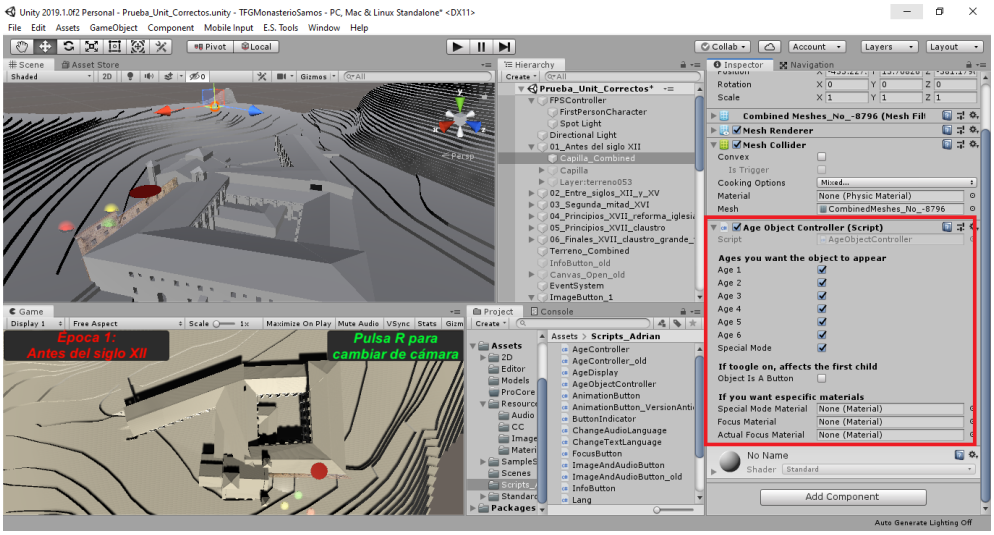


Figura 3.3: AgeDisplayController y su tag

3.4.3 AgeObjectController

Otros

El script más básico. Permite definir las épocas (Age) en las que el objeto aparecerá. Tambien se puede indicar si se quiere mostrar en el SpecialMode o no.

Si el objeto al que añadimos es un botón interacionable por el usuario (no de interfaz), debemos marcar la casilla correspondiente. Podemos añadir materiales específicos al objeto para el SpecialMode o para el FocusButton (scripts explicado más adelante). Si no introducimos nada, se tomarán los valores por defecto.

Para su correcto funcionamiento, las etiquetas ModeController y AgeController deben estar asignadas (Figura 3.4).

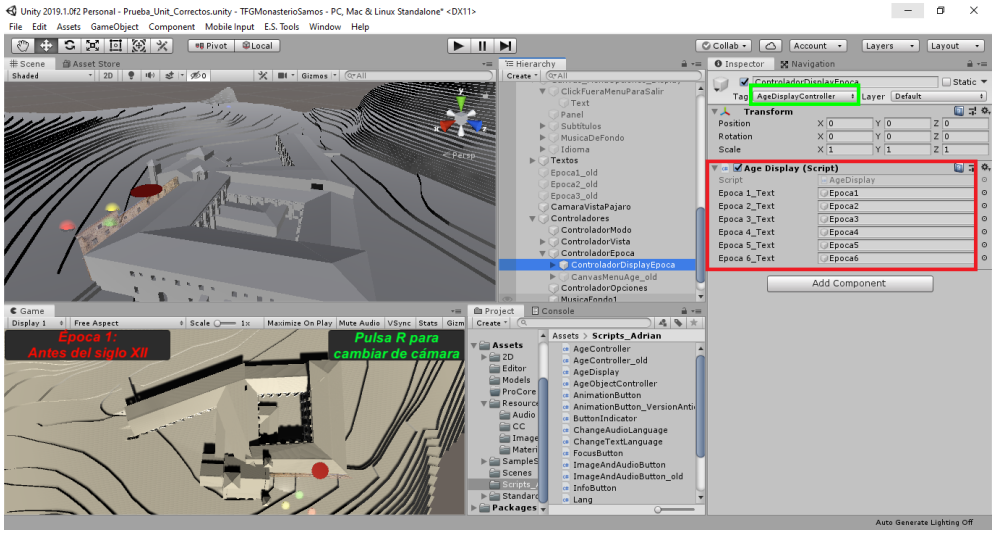


Figura 3.4: AgeDisplayController

3.4.4 AnimationButton

Iteración

Permite definir hasta 5 objetos distintos que se irán destacando según pulsemos la tecla de avance.

Debemos enlazar un texto, que aparecerá cuando nos acerquemos al botón.

Todos los objetos enlazados se verán con el material Focus la primera vez que pulsemos el botón. Si lo volvemos a pulsar, el primer objeto enlazado que tenga el atributo IsFocuseable activo se destacará entre el resto, apareciendo con el material ActualFocus. Si seguimos pulsando, cambiará de objeto destacado hasta que no quede ninguno.

Como todos los botones, este script deberá estar en el objeto primer hijo del Botón (que tendrá un AgeObjectController), acompañado de un component Collider para poder activar el botón.

Comprobar la configuración de la demo para más detalles (Figuras 3.5 y 3.6).

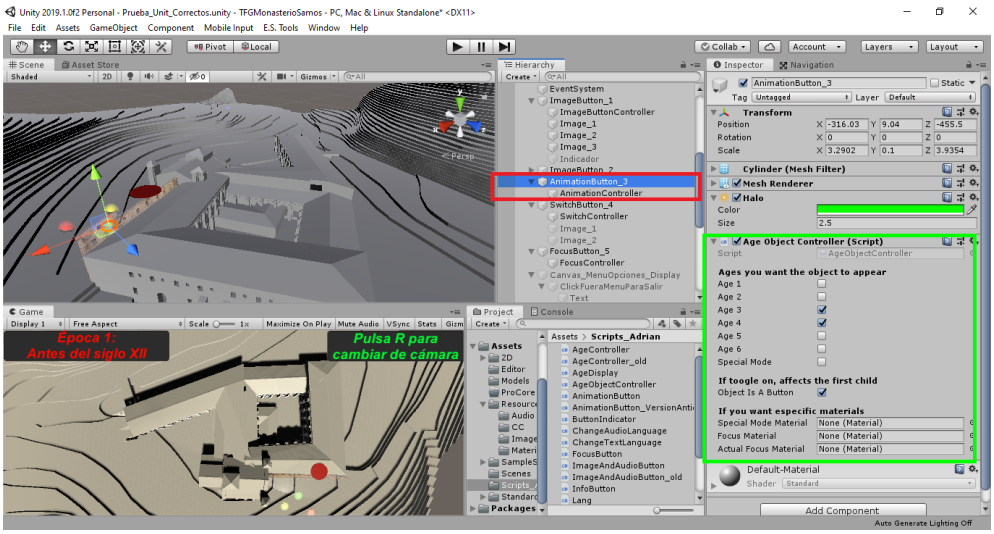


Figura 3.5: AnimationButton con AgeObjectController

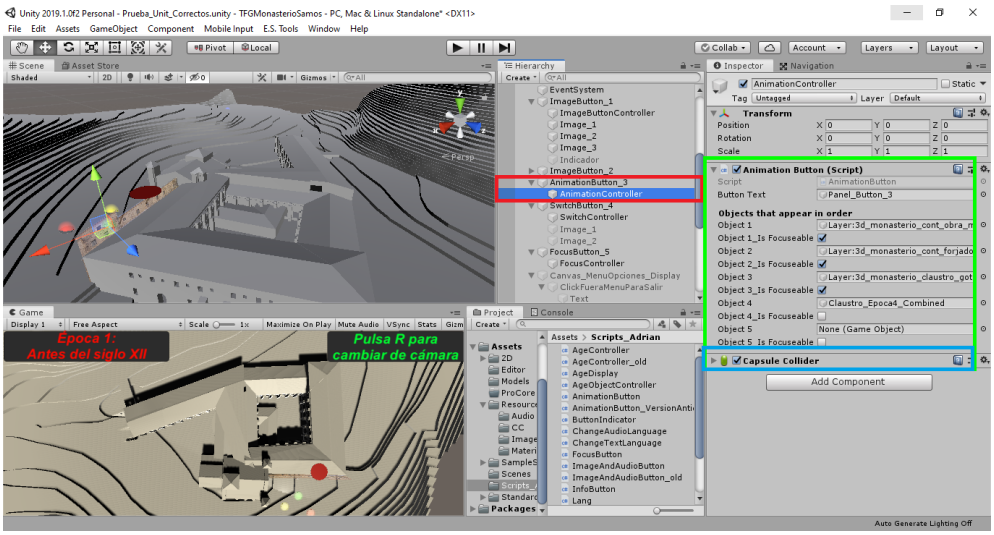


Figura 3.6: AnimationButton en el primer hijo

3.4.5 ChangeAudioLanguage

Otros

Permite cambiar el clip de audio que se esté utilizando por el correspondiente al idioma seleccionado. Suele ir acompañado de scripts con audio tales como FocusButton.

Enlazaremos el clip adecuado según el idioma.

Los objetos que tengan este componente deberán tener la etiqueta AudioLang (Figura 3.7).

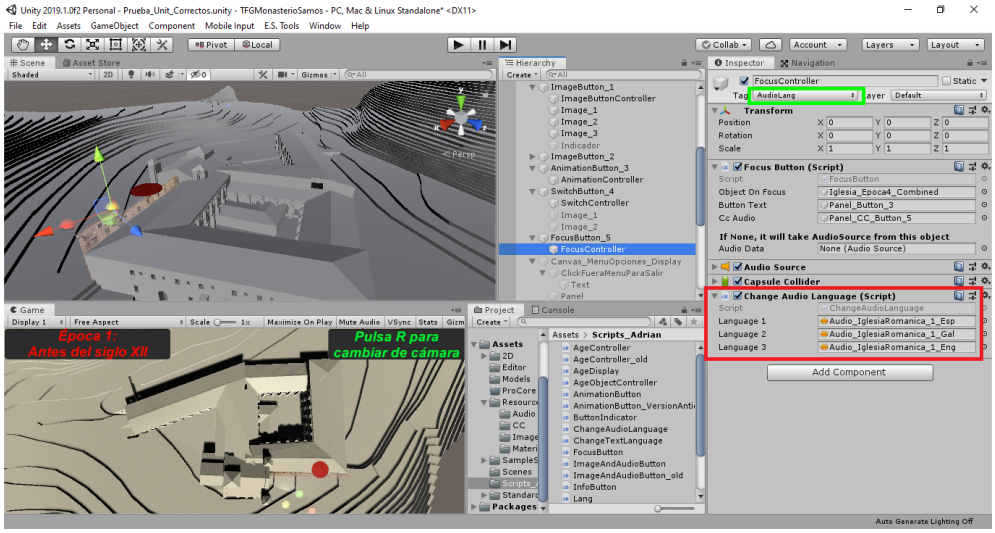


Figura 3.7: ChangeAudioLanguage y su tag

3.4.6 ChangeTextLanguage

Otros

Similar a ChangeAudioLanguage, permite cambiar un texto por el correspondiente al idioma activo. Suele ser usado por elementos con texto como los de la interfaz de usuario o los menús. Pondremos la referencia del atributo del archivo xml que contiene los textos de los idiomas (Figura 3.8).

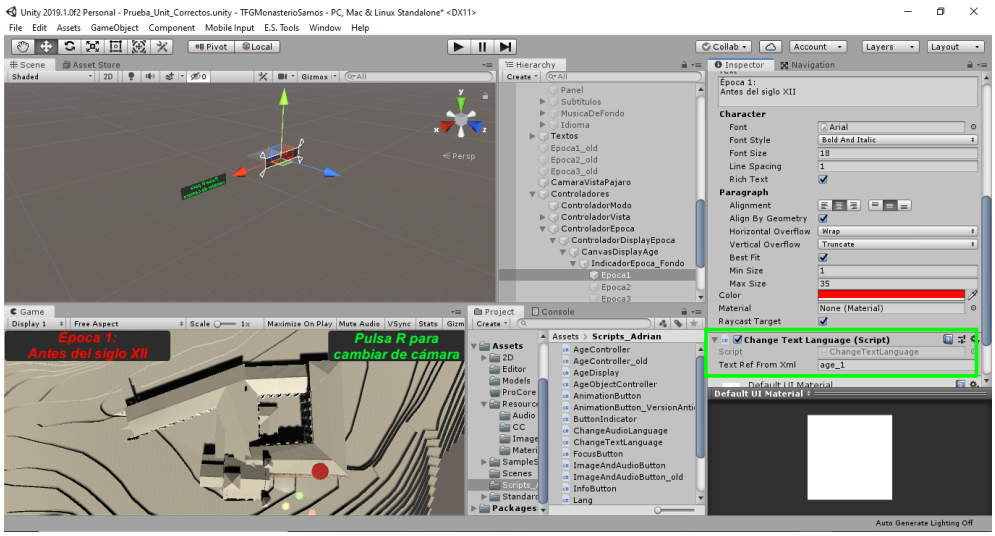


Figura 3.8: ChangeTextLanguage

3.4.7 FocusButton

Iteración

Permite enfocar la atención hacia un objeto en concreto (que aparecerá con su material de ActualFocus definido o, si no tiene uno asignado, por el material por defecto). A mayores, permite reproducir un clip de sonido y subtítulos.

Debemos enlazar el objeto a enfocar, el texto del botón, y los subtítulos en caso de que haya audio enlazado. El clip de audio deberá ir en el AudioSource del objeto. Se puede utilizar otro AudioSource, en cuyo caso habrá que enlazarlo en su correspondiente hueco.

Como todos los botones, este script deberá estar en el objeto primer hijo del Botón (que tendrá un AgeObjectController), acompañado de un component Collider para poder activar el botón. Comprobar la configuración de la demo para más detalles (Figura 3.9).

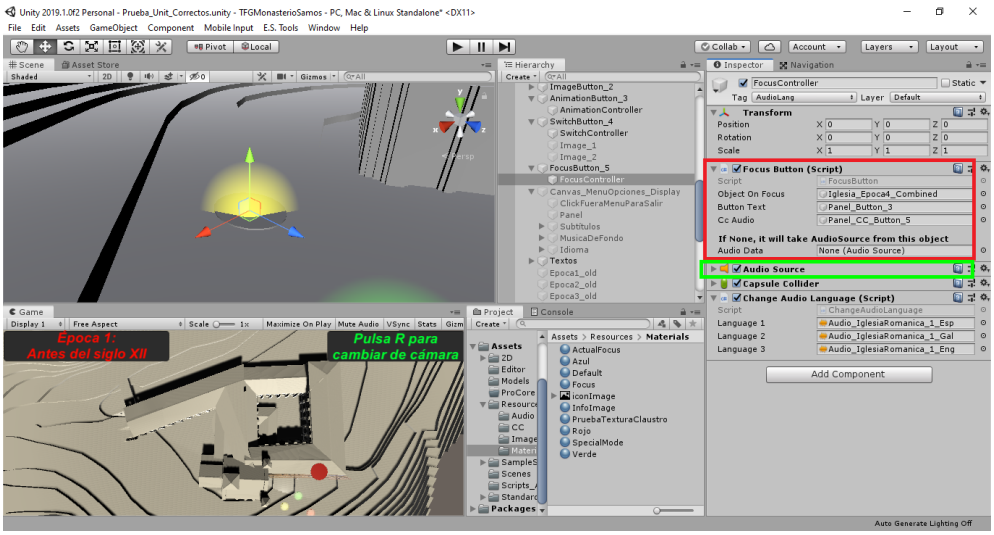


Figura 3.9: FocusButton y el AudioSource

3.4.8 ImageAndAudioButton

Iteración

Permite mostrar hasta 5 imagenes distintas, acompañadas opcionalmente de audio y subtítulos. Según pulsamos la tecla de acción, va cambiando entre las imágenes.

Debemos enlazar hasta 5 imágenes, el texto del botón, y los subtítulos.

El clip de audio deberá ir en el AudioSource del objeto. Se puede utilizar otro AudioSource, en cuyo caso habrá que enlazarlo en su correspondiente hueco.

Nota: en la imagen de muestra aparecen las imágenes como objetos hijo del botón, pero no es necesario. Pueden estar en cualquier parte de la jerarquía de objetos, siempre y cuando estén enlazados en el componente.

Como todos los botones, este script deberá estar en el objeto primer hijo del Botón (que tendrá un AgeObjectController), acompañado de un component Collider para poder activar el botón. Comprobar la configuración de la demo para más detalles (Figura 3.10).

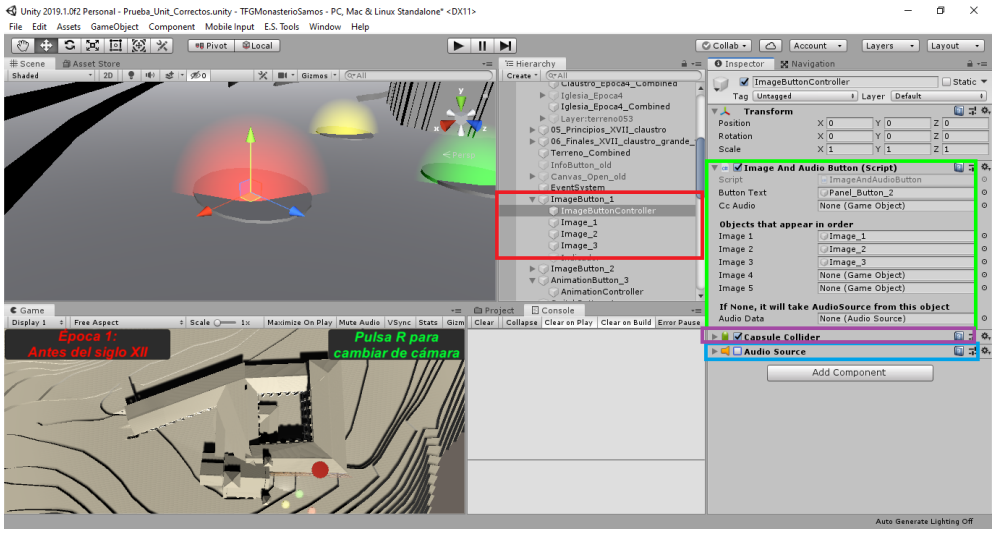


Figura 3.10: ImageAndAudio, los hijos y los componentes

3.4.9 Lang

Otros

Este script forma parte del conjunto de scripts relacionados con la selección de idioma. No es necesario que esté asociado como componente a ningún objeto, pero sí que esté presente en el proyecto, ya que es usado por otros scripts.

3.4.10 MapIndicator

Otros

Pequeño script que controla el indicador del Jugador cuando cambiamos a vista aérea.
El objeto que tenga este componente deberá tener la etiqueta PlayerIndicator.
Para su correcto funcionamiento, las etiquetas PlayerIndicator y Player deben estar asignadas (Figura 3.11).

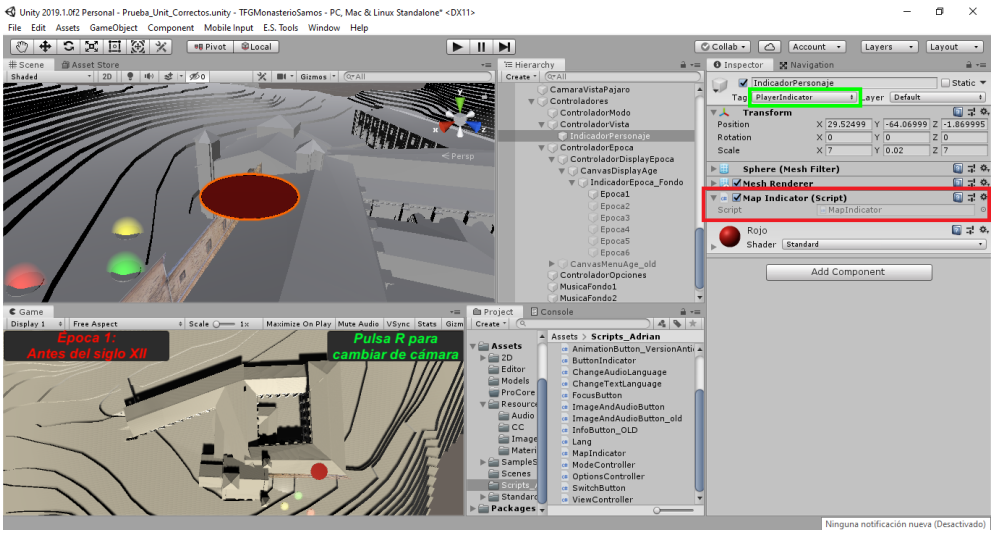


Figura 3.11: MapIndicator y su tag

3.4.11 ModeController

Controlador

Script que controla el cambio de modo (Normal o Viaje Temporal/SpecialMode).

Debemos enlazar el objeto Panel de texto que se mostrará en la interfaz cuando cambie el modo y, opcionalmente, un fondo para la pantalla que se mostrará durante el Special Mode.

El objeto que tenga este componente deberá tener la etiqueta ModeController.

Para su correcto funcionamiento, las etiquetas OptionsController y Player deben estar asignadas (Figura 3.12).

3.4.12 OptionsController

Controlador

Script que controla el menú de opciones y algunos ajustes generales, como el cambio de idioma. Podremos definir hasta dos audios que sonarán de fondo. Durante el SpecialMode, el segundo audio se desactivará.

Debemos enlazar el objeto del menú de opciones que contiene el cambio de idioma y el Canvas que contendrá toda la interfaz del menú de opciones.

El objeto que tenga este componente deberá tener la etiqueta OptionsController.

Para su correcto funcionamiento, la etiqueta Player debe estar asignada (Figura 3.13).

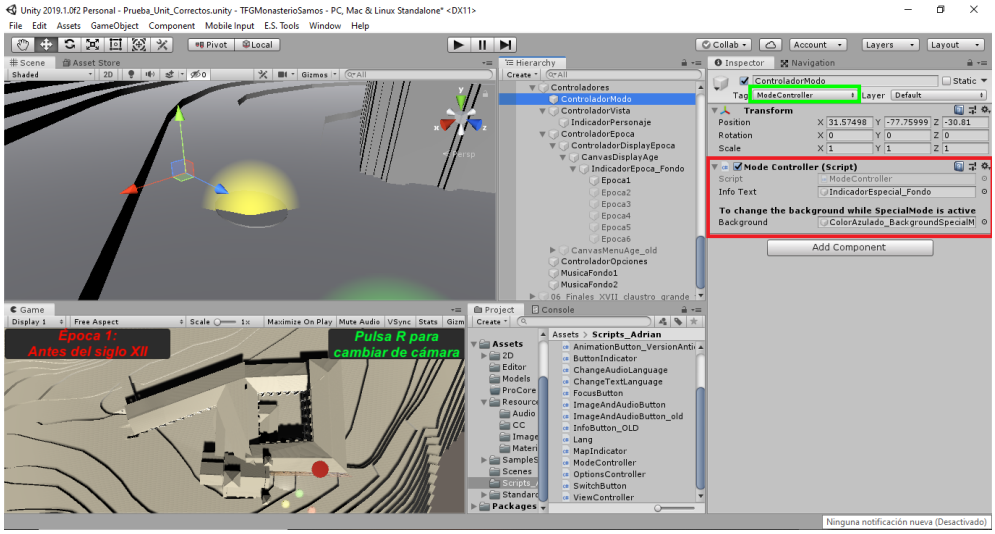


Figura 3.12: ModeController y su etiqueta

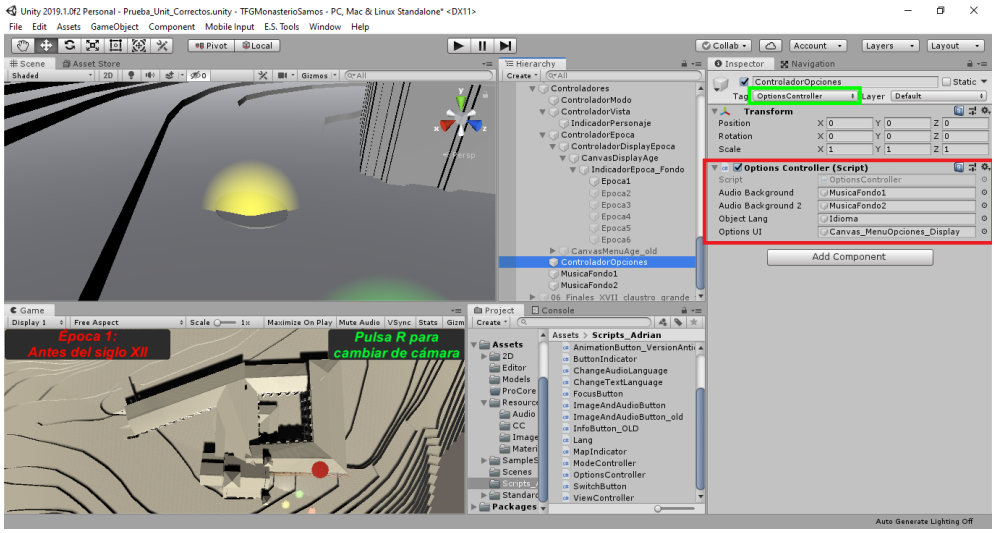


Figura 3.13: OptionsController y su etiqueta

3.4.13 SwitchButton

Iteración

Permite mostrar hasta 5 imágenes distintas, que no desaparecen si abandonamos la plataforma. Pulsando la tecla de acción, activamos y desactivamos su visión. Las imágenes desaparecen si cambiamos a una época donde el botón no está activo.

Debemos enlazar el texto del botón y podemos enlazar hasta 5 imágenes.

Nota: en la imagen de muestra aparecen las imágenes como objetos hijo del botón, pero no es necesario. Pueden estar en cualquier parte de la jerarquía de objetos, siempre y cuando

estén enlazados en el componente.

Como todos los botones, este script deberá estar en el objeto primer hijo del Botón (que tendrá un AgeObjectController), acompañado de un component Collider para poder activar el botón. Comprobar la configuración de la demo para más detalles (Figura 3.14).

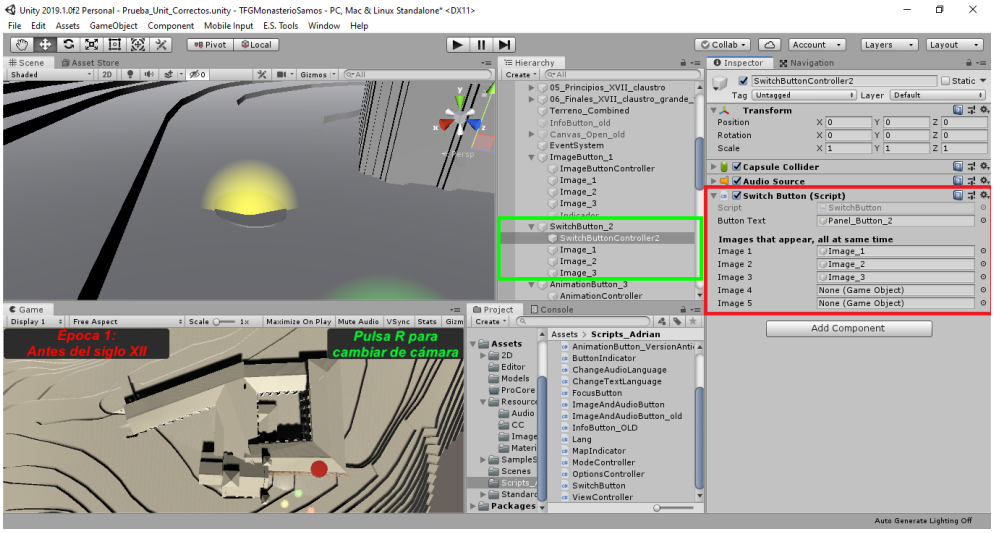


Figura 3.14: SwitchButton

3.4.14 ViewController

Controlador

Script que controla el cambio de cámara (1º persona / aérea). Debemos enlazar ambas cámaras y el texto que acompaña a la cámara aérea para volver a la cámara original.
El objeto que tenga este componente deberá tener la etiqueta ViewController.
Para su correcto funcionamiento, las etiqueta Player y PlayerLight deben estar asignadas (Figura 3.15).

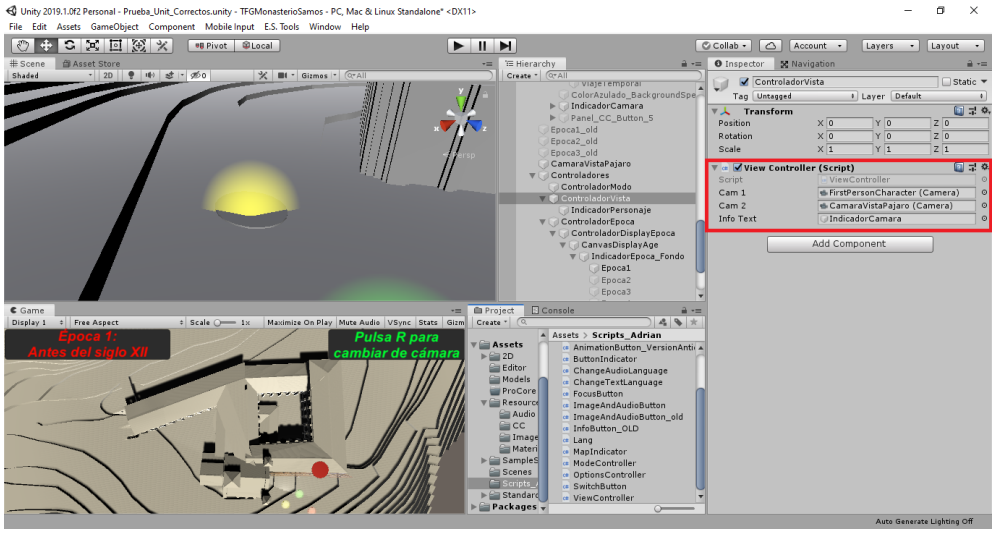
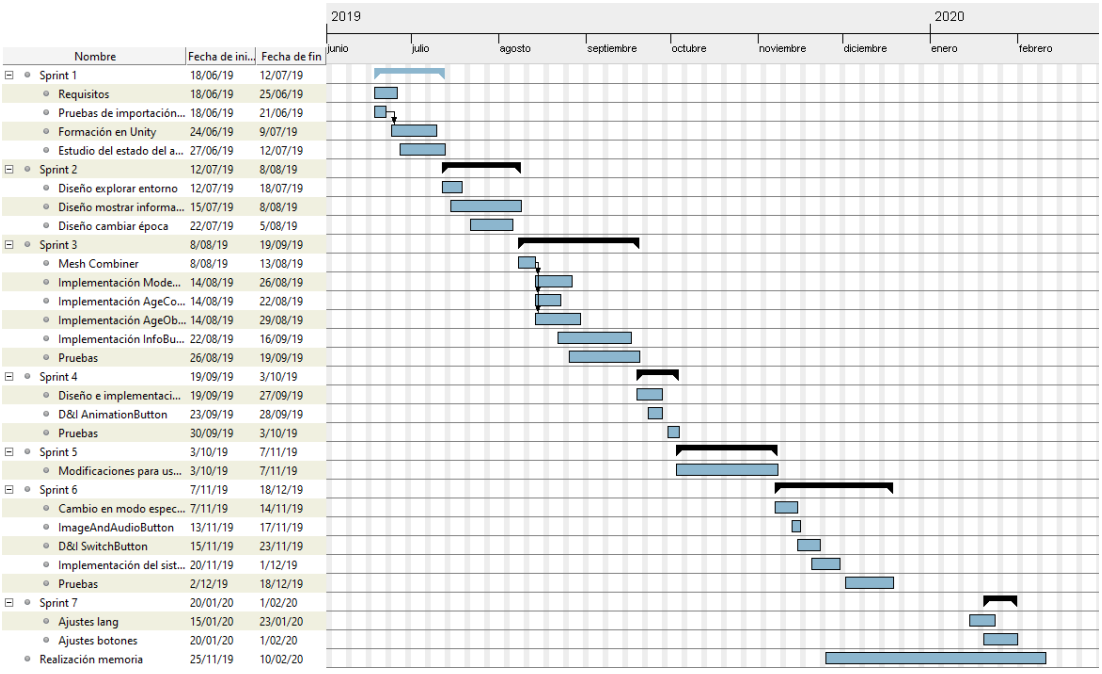


Figura 3.15: SwitchButton

Apéndice C

Diagrama completo de Gantt

En este apéndice se adjunta el diagrama de Gantt completo de las tareas planificadas del proyecto.



Nombre	Fecha de ini...	Fecha de fin
Sprint 1	18/06/19	12/07/19
Requisitos	18/06/19	25/06/19
Pruebas de importación...	18/06/19	21/06/19
Formación en Unity	24/06/19	9/07/19
Estudio del estado del a...	27/06/19	12/07/19
Sprint 2	12/07/19	8/08/19
Diseño explorar entorno	12/07/19	18/07/19
Diseño mostrar informa...	15/07/19	8/08/19
Diseño cambiar época	22/07/19	5/08/19
Sprint 3	8/08/19	19/09/19
Mesh Combiner	8/08/19	13/08/19
Implementación Mode...	14/08/19	26/08/19
Implementación AgeCo...	14/08/19	22/08/19
Implementación AgeOb...	14/08/19	29/08/19
Implementación InfoBu...	22/08/19	16/09/19
Pruebas	26/08/19	19/09/19
Sprint 4	19/09/19	3/10/19
Diseño e implementaci...	19/09/19	27/09/19
D&I AnimationButton	23/09/19	28/09/19
Pruebas	30/09/19	3/10/19
Sprint 5	3/10/19	7/11/19
Modificaciones para us...	3/10/19	7/11/19
Sprint 6	7/11/19	18/12/19
Cambio en modo espec...	7/11/19	14/11/19
ImageAndAudioButton	13/11/19	17/11/19
D&I SwitchButton	15/11/19	23/11/19
Implementación del sist...	20/11/19	1/12/19
Pruebas	2/12/19	18/12/19
Sprint 7	20/01/20	1/02/20
Ajustes lang	15/01/20	23/01/20
Ajustes botones	20/01/20	1/02/20
Realización memoria	25/11/19	10/02/20

Figura C.1: Diagrama completo de las tareas del proyecto. Creado con GanttProject

Lista de acrónimos

BIC *Bien de Interés Cultural*

CC *Closed Captions*: Subtítulos.

HUD *Heads-Up Display*: Información que permanece constantemente en pantalla, informando al usuario de elementos como tiempo, puntuación, salud, etc.

UI *User Interface*: Es el medio con que el usuario puede comunicarse con una máquina, equipo, computadora o dispositivo.

XML *eXtensible Markup Language*: Lenguaje de marcas extensible usado para almacenar datos de forma legible, usado extensamente para el intercambio de información entre aplicaciones.

Glosario

Mock-up: Modelo o prototipo que se utiliza para exhibir o probar un diseño. Se suele desarrollar para conocer la opinión de usuarios o consumidores, y suele tener implementadas sólo un número determinado de las funcionalidades del producto final.

Mesh: Colección de vértices, lados y caras que define la forma de un objeto 3D poliédrico. Las caras habitualmente suelen ser polígonos simples tales como triángulos o cuadrados.

Shader: Programa usado para la generación de efectos de sombreado y de postprocesamiento de imágenes o vídeo. Es el encargado de la producción de niveles apropiados de luz, oscuridad y color de una imagen.

Prefab: Objeto de Unity que funciona como plantilla, posee diferentes propiedades y componentes predefinidos que permiten crear varias instancias de un mismo objeto.

Renderizado: Proceso de crear una imagen para ser representada en pantalla.

Bibliografía

[1] "Web oficial de SpatioScholar," https://spatioscholar.njit.edu/.

[2] "Web oficial del UST," http://www.ust.ucla.edu/ustweb/ust.html.

[3] "Página oficial del videojuego Horizon Zero Dawn," https://www.guerrilla-games.com/play/horizon.

[4] "Web oficial de Unity," https://unity.com/es.

[5] "Web oficial de Blender," https://www.blender.org/.

[6] "Web oficial de Autodesk 3D MAX," https://www.autodesk.es/products/3ds-max/overview.

[7] "Web oficial de Notepad++," https://notepad-plus-plus.org/.

[8] "Web oficial de Unreal Engine," https://www.unrealengine.com/en-US/.

[9] "Web oficial de Source," https://developer.valvesoftware.com/wiki/Source_Engine_Features.

[10] B. Altın, "Psychiatry, space, and time: Case of an ottoman asylum," *Journal of the Ottoman and Turkish Studies Association*, vol. 5, p. 67, 04 2018.

[11] A. Wendell, B. O. Altin, and U. Thompson, "Prototyping a temporospatial simulation framework: Case of an ottoman insane asylum."

[12] A. Cockburn, *Agile Software Development*, 01 2002.

[13] A. A. G. CARMEN LASA GOMEZ, *MÉTODOS ÁGILES: SCRUM, KANBAN, LEAN*, 2nd ed. Anaya, 2017.

[14] E. López-Salas, "San julián de samos-lugo, estudio e interpretación del diseño monástico y su evolución," Ph.D. dissertation, 06 2015.

[15] B. Tan and H. Rahaman, "Virtual heritage : Reality and criticism," 01 2009, pp. 143–156.

[16] Gru, "Herramienta de uso libre complementaria a Unity," https://forum.unity.com/threads/mesh-combine-wizard-free-unity-tool-source-code.444483/.

[17] D. Staley, *Computers, Visualization, and History: How New Technology Will Transform Our Understanding of the Past*, 01 2002, vol. 31.

[18] Gru, "Script que permite el uso de múltiples idiomas en los proyectos de Unity," https://forum.unity.com/threads/add-multiple-language-support-to-your-unity-projects.206271/?_ga=2.147858999.1208944681.1575487851-1163745463.1555433089.

[19] C. Larman, *UML y patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado*, 2nd ed. Pearson, 2003.

[20] W. Duncan, "A guide to the project management body of knowledge (pmbok guide)," 01 2009.

[21] "BOE núm. 251, de 18 de octubre de 2019, pp. 114772 a 114801," https://www.boe.es/eli/es/res/2019/10/07/(8).